2012

Solving hard problems in election systems

Andrew Lin

Solving Hard Problems in Election Systems

by

Andrew Peter Lin

A dissertation submitted to Rochester Institute of Technology
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computing and Information Sciences

B. Thomas Golisano College of Computing and Information Sciences

Approved by Pengcheng Shi, Ph. D. Program Director

Signature Date

March 2012, Rochester, NY, USA

CERTIFICATE OF APPROVAL

DOCTORAL DISSERTATION

This is to certify that the Doctoral Dissertation of

Andrew Peter Lin

has been examined and approved by the dissertation committee as complete and satisfactory for the dissertation requirement for the degree of Doctor of Philosophy in Computing and Information Sciences

---

Dr. Edith Hemaspaandra
Advisor

Dr. Lane Hemaspaandra

Dr. Christopher Homan

Dr. Stanisław Radziszowski

Dr. Jeff Pelz
Chair, Dissertation Defense Committee

March 2012, Rochester, NY, USA

Abstract

An interesting problem in the field of computational social choice theory is that of elections, in which a winner or set of winners is to be deduced from preferences among a collection of agents, in a way that attempts to maximize the collective well-being of the agents. Besides their obvious use in political science, elections are also used computationally, such as in multiagent systems, in which different agents may have different beliefs and preferences and must reach an agreeable decision.

Because the purpose of voting is to gain an understanding of a collection of actual preferences, dishonesty in an election system is often harmful to the welfare of the voters as a whole. Different forms of dishonesty can be performed by the voters (manipulation), by an outside agent affecting the voters (bribery), or by the chair, or administrator, of an election (control). The Gibbard-Satterthwaite theorem shows that in all reasonable election systems, manipulation, or strategic voting, is always inevitable in some cases. Bartholdi, Tovey, and Trick counter by arguing that if finding such a manipulation is NP-hard, then manipulation by computationally-limited agents should not pose a significant threat. However, more recent work has exploited the fact that NP-hardness is only a worst-case measure of complexity, and has shown that some election systems that are NP-hard to manipulate may in fact be easy to manipulate under some reasonable assumptions.

We evaluate, both theoretically and empirically, the complexity, worst-case and otherwise, of manipulating, bribing, and controlling elections. Our focus is particularly on scoring protocols. In doing so, we gain an understanding of how these election systems work by discovering what makes manipulation, bribery, and control easy or hard. This allows us to discover the strengths and weaknesses of scoring protocols, and gain an understanding of what properties of election systems are desirable or undesirable.

One approach we have used to do this is relating the problems of interest in election systems to problems of known complexity, as well as to problems with known algorithms and heuristics, particularly Satisfiability and Partition. This approach can help us gain an understanding of computational social choice problems in which little is known about the complexity or potential algorithms. Among other results, we show how certain parameters and properties of scoring protocols can make elections easy or hard to manipulate. We find that the empirical complexity of manipulation in some cases have unusual behaviors for its complexity class. For example, it is found that in the case of manipulating the Borda election of unweighted voters with an unbounded candidate cardinality, the encoding of this problem to Satisfiability performs especially well near the boundary cases of this problem

and for unsatisfiable instances, both results contrary to the normal behavior of NP-complete problems.

Although attempts have been made to design fair election systems with certain properties, another dilemma that this has given rise to is the existence of election systems in which it is hard to elect the winners, at least in the worst case. Two notable election systems in which determining the winners are hard are Dodgson and Young. We evaluate the problem of finding the winners empirically, to extend these complexity results away from the worst case, and determine whether the worst-case complexity of these hard winner problems is truly a computational barrier. We find that, like most NP-complete problems such as Satisfiability, many instances of interest in finding winners of hard election systems are still relatively simple. We confirm that indeed, like Satisfiability, the hard worst-case results occur only in rare circumstances. We also find an interesting complexity disparity between the related problems of finding the Dodgson or Young score of a candidate, and that of finding the set of Dodgson or Young winners. Surprisingly, it appears empirically easier for one to find the set of all winners in a Dodgson or Young election than to score a single candidate in either election.

Acknowledgements

We wish to offer our special thanks to Curtis Menton, Christopher Connett, Dr. Edith Hemaspaandra and the other committee members Dr. Lane Hemaspaandra, Dr. Christopher Homan, Dr. Stanisław Radziszowski, and anonymous referees for their helpful comments. This thesis is supported in part by NSF grants IIS-0713061 and CCF-1101452.

Contents

Chapter 1

Introduction

Elections are a means for choosing one or more candidates given the preferences of individuals to arrive at a decision that attempts to maximize the collective welfare of the individuals (see also [BH01]). A voting system contains both rules for valid voting, and how one yields a final outcome given the preferences of individual voters. The choice of which rule one uses to choose the winner(s) may affect both the outcome of the election and the behavior of each voter. Voters may be unweighted or weighted, and need not count equally toward the final result.

The computational theory behind voting, as well as systems beyond that of the obvious majority and plurality, were first studied during the French Revolution. Two notable early theorists in this field were Jean-Charles de Borda [Bor81] and Marquis de Condorcet [Con85]. Borda introduced the system used for electing members to the French Academy of Sciences in 1770. The Borda system favors candidates that are somewhat liked by a lot of voters. Condorcet believed the winner should be one who fares well in pairwise comparisons of the candidates, a concept that is poorly captured by the Borda election, as it does not view pairs of candidates.

Condorcet discovered that, among voters with transitive preferences (meaning that if a voter prefers x to y and y to z, he or she will prefer x to z), majority preferences between pairs of candidates may be intransitive. This means that among candidates x, y, and z, it is possible for a majority of voters to prefer x to y, y to z, and z to x, and thus a Condorcet winner, who outperforms each alternative in a pairwise election, may not exist. The result that the pairwise preferences of the aggregate of rational voters may be irrational is known as the Condorcet paradox.

Beyond the applications in social sciences, in more recent times voting has also been

used in many interesting problems in computation theory, including rank aggregation in search engines [DNNS01, WH04] and collaborative filtering [GNO92]. Rank aggregation can also occur when one wishes to select documents or other results based on multiple criteria [LKRH90]. Some of these applications will be addressed in the next section.

Unfortunately, it is now known that no voting system is perfect, and three major results show the weaknesses inherent in voting. Arrow's impossibility theorem [Arr50] shows that in any election with as few as three candidates, no voting system can convert the ranked preferences of a group of voters into a community-wide ranking meeting a reasonable set of criteria: unrestricted domain, meaning the aggregation is deterministic and complete for all possible profiles of the voters, nondictatorship, meaning that no single voter should be able to single-handedly determine the outcome, Pareto efficiency, meaning that if every voter prefers a candidate to another, this order must also hold in the aggregation, and independence of irrelevant alternatives, meaning that changes to the voters' rankings of irrelevant candidates should not change the outcome for the candidate in question.

Beyond the problems of fairness, election systems are often subject to misuses, where the outcome of the election is unfairly affected by some of the voters, the chair of the election, or outside agents whom attempt to influence the voters. One form of misuse is when a voter reports preferences that are insincere to his or her true preferences for his or her own benefit, or when a group of voters conspire to collectively affect the outcome of the election. This is called manipulation, and is often disastrous for the system as a whole. In particular, a common problem in many election systems is the design that encourages voters to bury their 2^{nd} desired candidate, who may also be popular, among their preference list, giving their favorite candidate a more significant advantage. Done as a whole in the election, this distorts the true preferences of the participants. Two important results showing this weakness are the Gibbard-Satterthwaite theorem [Sat75, Gib73] and Duggan-Schwartz theorem [DS00], which show that eliminating the possibility of manipulation is impossible for any reasonable election system.

The Gibbard-Satterthwaite theorem [Sat75, Gib73] shows that every election of three or more candidates that chooses a single winner must either be dictatorial (i.e., where a single individual can choose the winner), nonsurjective (i.e., where some candidate cannot win under any circumstances), or is theoretically vulnerable to tactical voting in some cases. Tactical voting occurs when a voter or coalition of voters with full knowledge of the other voters' preferences has an incentive to vote contrary to their true preferences. Since neither of the first two properties are acceptable in any reasonable election system, this can be inter-

preted as all elections are subject to manipulation. The Duggan-Schwartz theorem [DS00] shows the same result with elections that elect a nonempty set of winners.

We illustrate the problem of manipulation with the following example. In the election system Veto, the lowest candidate in each preference profile is given one veto, and the candidate with the fewest vetoes wins. Consider a system of three candidates, x, y, and z, and, initially, 100 voters. 40 initial voters have preferences expressed by $y \succ z \succ x$, 35 have preferences $x \succ z \succ y$, and 25 have preferences $y \succ x \succ z$. A total of 22 manipulators have sincere preferences expressed by $x \succ y \succ z$. In an honest election, x would receive 40 vetoes, y 35 vetoes, and z 47 vetoes, making y the winner. In this case, the 22 manipulators can collectively make x the winner if 6 of the manipulators submit the preference $x \succ z \succ y$. Because x would not be the winner of an honest Veto election, such manipulations may produce a sub-optimal outcome. In this example, the manipulators need to know the preference orderings of the honest voters, which may or may not be true depending on the problem of interest. Unfortunately, all reasonable election systems have cases in which such voting behavior is encouraged.

In a related problem, an outside agent can also convince, or bribe, some of the agents to change their votes [FHH09]. In the computational problem of bribery, the goal of the briber is to modify the outcome of the election with the least amount of effort. There are many interesting problem models. Most notably, voters may be unpriced or priced. In the unpriced case, we are attempting to find a bribery which changes the preferences of the fewest number of voters, and in the priced case, each voter is given a price tag. In this case, we want to find the cheapest bribery. In other cases, the cost of bribing a voter may depend on how significantly his or her preferences are to be changed [FHH09] (see also [FHHR09]).

In the above example, we have a total of 122 unweighted voters, 40 of which choose x as their least favorite candidate, 35 choose y, and 47 choose z, making y a unique winner. We may make x a winner by convincing three voters whom dislike x to submit preference profile $z \succ x \succ y$ instead. This gives x a total of 37 vetoes, y 38 vetoes, and z 47 vetoes. This is an optimum solution if each voter is equally costly to bribe, as it bribes the fewest voters.

In other cases that we will evaluate, some voters are more costly to bribe than others. For example, in a campaign, it may be noted that a subgroup of voters will take more effort for one to convince. In the example above, depending on the price to bribe each voter, it may be the case that it is less costly to bribe five (six if a unique winner is desired) voters whom dislike z, to veto y instead. This gives x a total of 40 vetoes, y 40 vetoes, and z 42 vetoes.

It is further possible for the chair of an election to control the outcome of an election by manipulating the set of voters or candidates that will be involved [BTT92]. One such way to control is by encouraging or discouraging potential voters from participating. One prominent example in politics occurred in 1971 when President Nixon signed the 26^{th} amendment into the Constitution of the United States, lowering the legal voting age from 21 to 18, and thus adding a set of voters to the election. The chair can also partition the set of voters to modify the outcome of the election, similar to what is seen with electoral colleges in the United States. As often observed in the presidential elections, the winner of the popular (i.e., plurality) vote need not be the winner of the electoral vote. A related problem, in which some candidates are cloned (i.e., a new candidate with similar properties is introduced to split the voters of this candidate), has also been evaluated [EFS10] for several systems, including k-Approval. It is also possible to partition the set of candidates, as is done with the candidates of the Republican and Democratic parties, as well as to add or remove candidates.

In practice, an agent wishing to accomplish its goals will likely resort to a combination of more than one of the above strategies. Such problems are known computationally as multimode attacks [FHH11b, FHH11a], and will not be covered in this thesis.

Bartholdi, Tovey, and Trick [BTT89a] challenged the impossibility of avoiding election misuse by making the observation that manipulation only constitutes a threat when it is computationally feasible to determine how one may manipulate the election for a given instance in the system of interest. They considered an election to be computationally resistant to manipulation if determining such an exploit is NP-hard, and computationally vulnerable if it is polynomial-time computable, and proved a number of P and NP-hard results for manipulating some election systems with one strategic voter. Since then, many results have characterized this complexity result. For example, Conitzer, Lang, and Sandholm [CSL07] evaluate the model of manipulation by a coalition of voters. In this model, which we also use in this chapter, the coalition of voters are working together to affect the outcome of the election.

The line of reasoning of Conitzer, Lang, and Sandholm, of making coalitional manipulation at least NP-hard, was applied to bribery in [FHH09]. Hemaspaandra and Hemaspaandra [HH07] characterize these hardness results for the case of scoring protocols, by showing exactly which scoring protocols are computationally resistant or vulnerable to manipulation.

There also exist other models of interest for the problem of election manipulation, which we will not review in this thesis. In some models, each manipulator is not aware of the existence or actions of other potential manipulators. Models of manipulation under limited

communications in these cases include that of safe manipulation [SW08] (see also [IYEW11]). In this case, the manipulator wishes to vote in a way such that the election never produces an undesirable outcome. Other possible models involve probabilistic measures of susceptibility to manipulation [RPW11].

Another dilemma facing election theory is that some election systems believed to have desirable properties are provably difficult to evaluate, at least in the worst case [BTT89b, HHR97, RSV03]. Two prominent examples are Dodgson [Dod76] and Young [You77]. Both of these election systems attempt to capture the properties of Condorcet winners [Con85], by electing a Condorcet winner when one exists, and finding an "approximate" Condorcet winner otherwise. In both cases, it is NP-hard to find the score of a candidate, and Θ_2^p-complete to find the winners of an election.

A major weakness of any result showing NP-hardness, including that of problems in election systems, is the fact that NP-hardness only tells one that the problem is difficult in the worst case. It may not reflect the intractability of the cases one is interested in. Two important results in this area include that of Friedgut, Kalai, and Nisan [FKN08], and that of Walsh [Wal09]. In [FKN08], a quantitative version of the Gibbard-Satterthwaite theorem is shown, demonstrating a tradeoff between the closeness of an election system to that of a dictatorship, and the manipulation power of voters of the election, defined by the probability that a voter can change the outcome of the election by casting a random vote. Using probability theory, it is shown that for any election system reasonably far from being a dictatorship, there is a nonnegligible probability of a random change of a single voter's preferences causing a change in the outcome of the election. This is true even in the region of the problem space known as the phase transition. It is found that in many other problems, hard instances lie within the phase transition of the problem [GW96], where the problem is neither trivially underconstrainted nor overconstrained. It is shown by Walsh that the manipulation problem of interest seems to avoid hardness even within this region.

The weakness of NP-hardness results applying only to worst-case complexities is also a prominent issue to the problem of finding the Dodgson score of a candidate in an election. It is shown in [HH06] that an obvious greedy algorithm for finding Dodgson scores is "almost always" self-knowingly correct (i.e., one can identify when the greedy algorithm is correct) when the voter set is much larger than the candidate set. In some cases, the approximation to an NP-hard problem can itself be provably useful. A prominent example also occurs in Dodgson elections [CCF+09], in which a randomized approximation of the Dodgson election has desirable election properties, lending its use as an election system itself.

Walsh [Wal09] takes the approach of evaluating the runtime complexity of finding a manipulation (or showing none exist) by invoking the Complete Karmarkar-Karp (CKK) Algorithm [Kor98] for the NP-complete problem of Partition. Walsh evaluates the case of the Veto election of three candidates, but states that similar reductions can be applied to other scoring protocols of three candidates, as well as the Veto election for four or more candidates (using Multi-Way Partition). By reducing instances of this problem of manipulation to instances of Partition, and evaluating these instances using the CKK algorithm, Walsh showed, experimentally, that instances in which it is hard to find a manipulation or show that one does not exist using this algorithm are exceedingly rare. It is further demonstrated that such instances occur when the voter weights and preferences are highly correlated, and that even one uncorrelated voter likely makes the manipulation problem empirically easy. Using some theoretical reasoning based on some known complexity results of the CKK algorithm for Partition [KKLO86], Walsh also makes use of probability theory, in conjunction to empirical results known about Partition and the CKK algorithm, to show why these hard instances are exponentially rare. In another result, Walsh makes use of other NP-complete problems and algorithms to show a similar result for a multi-round election system, Single Transferable Vote (STV) [Wal10].

NP-complete manipulation problems can also be evaluated using algorithms for other NP-complete problems, particularly Satisfiability. Satisfiability, or SAT, is the problem of determining whether a boolean formula can be satisfied with some assignment of its variables. The NP-completeness of SAT was established in the Cook-Levin theorem [Coo71, Lev73], and was the first decision problem proven to be NP-complete. Despite its NP-completeness, algorithms exist to solve SAT that are surprisingly practical for many problems of interest. Almost all algorithms for SAT involve a systematic backtracking search procedure to explore the exponential search tree of variable assignments to look for a satisfying assignment. The basis for such algorithms is the Davis-Putnam-Logemann-Loveland algorithm (DPLL) [DP60, DLL62], which operates recursively by assigning variables sequentially. Although all known algorithms for SAT have exponential-time worst-case complexity, different heuristics in the search have proven effective for solving SAT instances effectively depending on the application. Notable improvements in the DPLL algorithm have included backjumping [Pro93], which backtracks multiple levels of recursion whenever possible, and clause learning [BKS04], which involves adding additional constraints whenever an inconsistency is found. Both of these improvements work by reducing the search space. The application of SAT solvers in manipulating some elections is discussed in [Con10], in which it is used to determine the

minimum-size coalition needed to manipulate an election. However, no attempt is made in evaluating the complexity of doing so or the tradeoffs in different solvers and instance representations.

It is important to understand the known limitations of heuristics to NP-hard problems under our current assumptions of complexity theory. For example, it is known that no polynomial-time heuristic can correctly solve an NP-hard problem on all but a sparse (i.e., at most a polynomial number of strings of each length) set of inputs unless P=NP (see [Sch86]). It is further not possible to solve an NP-hard problem on all but a sub-exponential number of strings of each given length (i.e., $n^{\log^{\Theta(1)} n}$ strings of length n) in polynomial time unless EXP=NEXP (see [FHH11b]).

1.1 Applications of Election Systems

Interesting applications of elections in computational theory include rank aggregation and collaborative filtering [GNO92]. Rank aggregation may occur in search engines [DNNS01, WH04], or more generally when one wishes to select documents or other results based on multiple criteria [LKRH90]. In these cases, the multiple criteria or search engines may be considered opposing opinions, which can conceptually be represented by voters wishing to arrive at a common decision, the result of the search query.

Voting in this context is interesting because it is seldom that a single preference produced by a search algorithm will be reliable and unbiased. Search engines may be vulnerable to spam, for instance, in which the authors of Web pages may exploit the algorithm used by an individual engine to maliciously produce unreasonable rankings for itself. Methods of spamming performed by Web pages may include adding irrelevant text, adding erroneous inbound links, and cloaking, in which entirely different content is sent to the search engine crawler than to the users (see [HMS03]). Biased financial interests and other outside influences may also cause individual search engines to be unreliable. It is not unusual for paid placements to occur in search engines, for instance. Beyond undesirable biases, other problems include the search engine being broadly acceptable and comprehensive in its coverage, which may not be a practical assumption (see http://www.searchenginewatch.com/ for a survey on these problems in actual search engines). For these reasons, a very important problem in this field is for one to computationally aggregate the preferences of multiple search engines in a way to provide users with a robust search result, and in essence, filter out "noise" among the rankings and improve its overall properties, while maintaining fairness.

Some of the challenges apparent in aggregating rankings of search engines demonstrate the tradeoffs inherently involved in election systems. In an ideal world, we would wish for each individual search engine to give a complete ordering of all alternatives of interest, in this case, the set of all pages that are available to search. This is clearly impractical due to size. In addition, many search engines limit access of results to relatively few results to preserve the confidentiality of the search algorithm, and for efficiency purposes.

As it turns out, even if it were practical to obtain complete and honest information from the individual agents, many interesting systems of aggregation with proven robustness results are also exceedingly difficult to evaluate. One aggregation function of particular interest is the Kemeny optimal aggregation [YL78], which is based on the model that there is one correct preference order, and that the individual preferences of the voters are simply noisy approximations of this order. The Kemeny optimal aggregation is the ordering which minimizes the total Kendall Tau distance, which measures distances between rankings. This system is interesting because it satisfies a number of fairness results, such as the Condorcet criterion [Con85]. Unfortunately, the Kemeny optimal aggregation is Θ_2^p-complete to compute [HSV05] for elections of four or more candidates. Also troubling is that in the case of building meta-search engines, there are often many candidates (pages) and few voters (search engines). This has the effect of further rendering this computation intractable. (We will look further at some systems in which problems in the complexity class of Θ_2^p arise and possible approximations and heuristics of these problems in Chapter 6.)

Due to the competing issues above, one needs, in many cases, to settle for approximations which may be less robust in satisfying fairness and hardness of manipulation properties, but may be computed in a reasonable amount of time. There are many interesting approximations, including algorithms involving approximations of the Kendall Tau distance that are easy to compute, and genetic algorithms for approximating the Kemeny ordering itself. It is also shown in [DNNS01] that the Condorcet criterion, and several variations thereof, also have important implications in the reduction of spam in search engine aggregation, and approximations satisfying these conditions is of interest. Evaluating each of these tradeoffs in efficiency and robustness is a problem of interest in election systems.

Search engine aggregation techniques also have applications in word association techniques and for evaluating the quality of a search engine. In particular, a search engine can be found to be of high quality if it is noted to behave close to that of the aggregation found.

Another notable computational application of voting is that of collaborative filtering, which, as its name implies, is the process of filtering for information from large pieces of data

using techniques involving collaboration among multiple agents. For example, one may wish to collect tastes from users for marketing purposes. A database of opinions obtained this way is invaluable in making future predictions of user opinion on additional items using heuristics. This application is based on the idea that two individuals with similar preferences in the past may have similar preferences in the future. Data that may be evaluated using collaborative filtering include sensing and monitoring data, such as in exploration and environmental sciences, financial data from institutions, or user data used for marketing purposes, such as what ads to show a particular user to maximize profit. Although an early model, Tapestry [GNO92] required explicit user action to obtain preference information, advances in this area have also included the ability to provide predictions with little or no user effort [THA+97]. In all of these cases, it is interesting to combine decisions from multiple sources each using different forms of judgement (see also [GSK+99]). All of these applications demonstrate the strong tradeoffs between tractability, fairness, and robustness against misuse, which is at the heart of the study of computational social choice theory.

1.2 Organization of the Thesis

In Chapter 2, we introduce terminology for election systems, define problems of interest in election systems, and also related problems of interest, including some variations thereof that we will use and known algorithms for these problems. We also review the relevant previous work in these problems. Throughout the rest of this thesis, we will be using some of these related problems and their algorithms to solve the problems of interest in election systems, as well as proving the difficulty of doing so.

In Chapter 3, we address the worst-case complexity of manipulating one of the simplest families of scoring protocols, that of $f(m)$ approval, where each candidate approves of some function $0 \leq f(m) \leq m$ of the m candidates in the election. We determine when such elections are easy or hard to control, manipulate, and bribe. Essentially, we extend the work of Hemaspaandra and Hemaspaandra [HH07] to cases of families of approval-based scoring protocols, in which the candidate set cardinality is unbounded. We also generalize the work of Bartholdi, Tovey, and Trick [BTT92], on control by adding and deleting candidates. We show how one can modify the construction to show hardness in more general cases. A characterization of an infinite set of election systems was also evaluated in [FHS08].

We find that there are generally only a few cases for the complexity of manipulation in such election systems: Nearly all problems of misuses of election systems involving controlling

the voter set are hard by reductions from Set-Cover-type problems, while some problems involving controlling the candidate set are hard by a reduction from Hitting Set. Other cases are easy by greedy algorithms or variations of the Edge Cover problem. We hope that from our work one can gain a better understanding of what properties of elections make them computationally resistant or vulnerable to manipulation, so that our work can be extended to more general forms of elections.

Because NP-hardness is only a measure of worst-case complexity and does not directly imply the infeasibility of solving the problem in the setting of interest, in Chapters 4, 5, and 6, we evaluate the complexity of some NP-hard problems in election manipulation in relationship to some known NP-hard problems and their solutions.

In Chapter 4, we extend the work of Walsh [Wal09], which shows that scoring protocols of three candidates are rarely difficult to manipulate using algorithms for Partition, to that of scoring protocols of more than three candidates. We do so by using known algorithms for k-Way Partition [Kor98, Kor09, Kor10]. Manipulation and Partition are related problems because a common issue in manipulating to the benefit of a candidate is the division of votes to the candidates not being manipulated for, to avoid electing them.

We show that Walsh's statement that the results of empirical simplicity hold for veto elections of more than three candidates do not follow directly from the problem of k-Way Partition, but require a nontrivial modification to this problem. We do, however, confirm Walsh's claim that these elections are equally easy to manipulate, using algorithms for the modified problem. Our new problem also allows one to evaluate manipulation problem in k-candidate scoring protocols generally. We empirically evaluate some of these cases. These new algorithms may also help us gain an understanding of the problem of Partition, due to the closeness of the problems of Partition and manipulation.

In Chapter 5, we evaluate the feasibility of solving manipulation problems using the known algorithms for Satisfiability (SAT), including some extended variants of SAT [DP60, DLL62, Pro93, BKS04, GKS10, ES06]. SAT is a common tool for the solving of NP-complete problems. A general reduction from election manipulation, for many systems including scoring protocols, to 0-1 Integer Linear Programming (ILP) is also evaluated by Connett [Con10]. 0-1 ILP can be seen as a generalization to the problem of SAT, and can often easily be encoded into SAT itself.

We demonstrate how to improve these reductions for some specific cases by exploiting symmetry in some scoring protocols, and show how this is crucial in showing the ease of manipulation for some cases. Most notably, in cases such as those involving unweighted

manipulators or cases in which candidates may be permuted in parts of the preference profiles (e.g., k-approval elections), a naive representation of the manipulation into variables of SAT may cause many similar manipulations to be represented by a large number of instantiations of the variables. This reduces the efficiency of invoking the algorithms of SAT.

Our results confirm the results of Walsh [Wal09] for scoring protocols of more than three candidates, showing that instances in which deciding manipulability is hard are rare. However, we find that these encodings, along with the algorithms for SAT used, do not extend the results of Walsh to families of scoring protocols, in which the candidate set cardinality is unbounded. We believe that a large candidate set may be a strong defense against manipulation, at least under the current understanding of NP-hardness.

We also find a number of surprising unintuitive behavior of the complexity of this encoding in some cases. In particular, in the case of manipulating the family of scoring protocols Borda on unweighted voters, it is found that the problem appears to be easier to solve near the boundary of manipulability, when we are given a number of manipulators close to the minimal coalition needed, and that unsatisfiable instances appear easier than satisfiable ones. Both of these observations are contrary to most NP-complete problems. Using an approximation algorithm of this problem [ZPR09], we find that these unusual properties allow us to find a manipulation of minimal size efficiently in all but an exceedingly rare subset of cases, despite NP-completeness [DKNW11, BNW09] and despite the unboundedness of the candidate set cardinality. Aside from the case of unweighted Borda, we find that almost all families of scoring protocols resist the current state-of-the-art in SAT solving on this encoding.

In Chapter 6, we evaluate the feasibility of scoring candidates and finding winners in the Condorcet-consistent election systems of Dodgson and Young. The problems of finding winners in Dodgson and Young elections are Θ_2^p-complete [HHR97, RSV03]. A problem in Θ_2^p can be solved in polynomial time given $O(\log n)$ queries to an NP oracle for problems of size n. The set of problems that are Θ_2^p-hard, that is, polynomial-time reducible to each problem in Θ_2^p was first studied by Papadimitriou and Zachos [PZ83] and Wagner [Wag90]. It has been established that the problems of Θ_2^p can also be characterized as problems that can be solved in polynomial time given parallel truth-table access to NP [Hem89, KSW87].

Reductions of problems in finding Dodgson and Young scores to 0-1 Integer Linear Programming, which is closely related to and can be converted into SAT, has been explicitly given in [BTT89b, CCF$^+$09], and, as in Chapter 5, we will make use of symmetry breaking techniques. In doing so, we extend the phase transition results of Walsh [Wal09] and demonstrate how the concept of phase transition appears to extend to problems outside of

NP-completeness. We also demonstrate how to efficiently query the NP-complete problem of scoring Dodgson and Young elections to find the winners of the election.

Our experimental results suggest that finding the Dodgson and Young winners is often easy, and the results are akin to [Wal09], which showed that instances of veto elections in which manipulations are hard to find are exceedingly rare. Surprisingly, the same underlying encoding did not show that the Dodgson and Young scores of candidates, a seemingly easier problem, are equally easy to compute. This is because it is seldom necessary to find the Dodgson and Young scores of all of the candidates in order for one to determine the set of winners. This result has some interesting implications on the applicability of the known complexity and approximation results of the scoring problems, if the ultimate goal is only to produce the set of winners.

By giving an algorithm for finding Dodgson and Young winners, we have also given an explicit algorithm to perhaps two of the best-known natural Θ_2^p-complete problems and shown, empirically, the phase transition of problems in this complexity class. The property of phase transition is more commonly seen in NP-complete problems [GW96, Wal09].

The results of Chapter 3 have appeared in the 2011 International Conference on Agents and Artificial Intelligence [Lin11a], and the results of Chapter 4 in the 2011 International Conference on Autonomous Agents and Multiagent Systems [Lin11b].

Chapter 2

Preliminaries

In this chapter, we define what an election system is, and formally define the problems that may occur in the system: manipulation, control, bribery, and winner problems. We then define other computational problems, such as Edge Cover, Partition, SAT, and 0-1 Integer Linear Programming, which will be utilized in evaluating the complexity of these election problems. Finally, we will review previous complexity results of these related computational problems.

2.1 Definition of Elections

An election $E = (C, V)$ is defined as a pair of a set of voters $V = \{v_1, \ldots, v_n\}$ and a set of candidates $C = \{c_1, \ldots, c_m\}$. Each voter v_i has a preference over the candidates. A common model for understanding voter preferences is that of the transitive voter model. In the transitive voter model, which we will use exclusively in this chapter, each preference is a strict linear ordering over the candidates $c_{i_1} \succ \cdots \succ c_{i_m}$. But voter preferences need not be transitive (see, e.g., [FHHR09], in which the voter submits a reflexive and antisymmetric binary ordering over the candidates C). Furthermore, voters in some election systems cannot submit their vote simply as a preference ordering. A prominent example is approval, which we will define below.

Elections may be weighted or unweighted. In a weighted election, each voter v_i has a weight, $w(v_i)$, and their vote is counted as $w(v_i)$ individual votes in an unweighted election with the same preference ordering.

2.1.1 Election Systems of Interest

An election system $\mathcal{E}$ specifies how one arrives at the outcome given the collection of voter preferences. Depending on the context, the outcome we may be interested in may be a winner, a nonempty subset of winners, or an aggregate preference ordering. Most typically, an election for us will be what in the literature is called a social choice correspondence, namely, a mapping from the candidates and the voter preferences to a subset (known as the winner set) of the set of candidates. There are many systems of interest, some of which involve multiple rounds. Perhaps the most common election system in everyday use is that of plurality, in which the candidate(s) most frequently ranked first among the voters is elected. Plurality is an election system in a more general family of election systems called scoring protocols.

A *scoring protocol* (see the handbook article [BF02]) is defined over a vector $(\alpha_1, \ldots, \alpha_m) \in \mathrm{Z}^m$. Each candidate c is given α_i points for each voter that ranks c in the i^{th} position of his or her ranking. The candidate(s) with the highest score wins. In addition to plurality, common scoring protocols include veto, Borda count, and approval-based systems with a fixed number of candidates.

A *family of scoring protocols* is an infinite series of scoring protocols $(\alpha^1, \ldots, \alpha^m, \ldots)$, where $\alpha^m = (\alpha_1^m, \ldots, \alpha_m^m)$ is a scoring protocol of m candidates.

In *plurality*, a common form of elections in political science, each voter gives one point to his or her favorite candidate, whereas in *veto*, each voter approves, or gives one point, to all but one candidate: This has the effect of vetoing one candidate. Plurality and veto can thus be considered a family of scoring protocols of the form $\alpha^m = (1, 0, \ldots, 0)$ and $\alpha^m = (1, \ldots, 1, 0)$ respectively.

In *k-approval*, each voter gives his or her k favorite candidates 1 point. Similarly, in *k-veto*, each voter vetoes his or her least k favorite candidates.

Although not a scoring protocol, another common election system is *approval* [BF78]. In approval voting, each voter can approve as many or as few candidates as he/she chooses. In some models, voters may still have a preference order, along with the number of candidates approved. The winner(s) is thus the candidate(s) with the most number of approvals (or weights thereof). In [FHH09, HHR07], it is shown that approval is computationally resistant to many forms of misuse involving the voter set, even with unweighted voters, for constructive and destructive cases. We will in fact examine the properties of this election system that give rise to the resistance.

A generalization of k-approval and k-veto, *$f(m)$-approval*, where f is a function of the number of candidates m, is an election where each voter gives 1 point to each of his or her $f(m)$ favorite candidates. In this case, we will assume that $f(m)$ may be computed in time polynomial with respect to the quantity m. This ensures we may compute the outcome of an instance of this election in polynomial time.

Another interesting scoring protocol is the *Borda count* [Bor81]. For m candidates, this election is the scoring protocol defined by the vector $(m-1, m-2, \ldots, 1, 0)$. In this case, the first preference of each voter is given $m-1$ points, the second $m-2$ points, and so forth.

A *Condorcet winner* [Con85] of an election is a candidate who beats each other candidate in a pairwise election. A candidate beats another candidate in a pairwise election if a majority of voters prefer the former to the latter; in other words, if the voters as a whole prefer this candidate to each other alternative. Similarly, a *weak Condorcet winner* is a candidate that is preferred to each other candidate by at least half of the voters.

Despite the simple definition, a Condorcet or weak Condorcet winner does not always exist in an election, even of three candidates. Because Condorcet winners have important properties, several systems have been suggested to extend the Condorcet election to a deterministic election system. An election system is said to be Condorcet-consistent if it elects the Condorcet winner when one exists. Many interesting election systems, such as that of Borda, are not Condorcet-consistent [Nan82].

Two notable Condorcet-consistent elections are the Dodgson and Young elections. In both cases, the election system attempts to find an approximate Condorcet winner when one does not exist by modifying the preference profile. The winner is then the candidate that can be made a Condorcet winner with the least amount of modification.

In the *Dodgson election* [Dod76], each candidate is given a score equal to the minimum number of exchanges between adjacent candidates in a voter's preference listing to make this candidate a Condorcet winner. The candidate with the lowest score, that is, needing the fewest such exchanges, wins. In essence, a Dodgson winner is an approximate Condorcet winner in the sense that it can be made a Condorcet winner with the fewest number of improvements within the preferences of each voter. The *Young election* [You77, RSV03] scores each candidate by the fewest number of voters that must be removed from the election to make the candidate in question a Condorcet winner. Young elections are essentially always defined in the literature (initially by Young himself [You77]) as the fewest number of voters that must be removed from the election to make the candidate a weak Condorcet winner. However, the paper [RSV03], apparently accidentally, used the term "Young elections" to

refer to the (corrupted) definition in which one seeks to make the candidate a Condorcet winner. That notion, a perfectly natural one to study, should most properly be referred to as "strong Young elections," to avoid any confusion as to what "Young elections" mean (see [BBHH10] for a further discussion of this potential confusion in the definition of Young elections). However, since strong Young elections are the only notion of Young-like elections that we use in this thesis (and we mention that this choice means that we are focusing on the same election system that [RSV03] focus on), we in an abuse of standard terminology will for the remainder of this thesis mean "strong Young election" each time we say "Young election." In each case, the candidate with the lowest Young score (i.e., it is the Condorcet winner among a large subset of voters) wins the Young election.

In both the Dodgson and Young systems, it is NP-hard for one to determine the score of a candidate [BTT89b, RSV03], and Θ_2^p-hard for one to determine the winner of an election [HHR97, RSV03]. Because the Condorcet-consistent criteria is an important measure of fairness, previous work has included approximations of the Dodgson and Young score and the properties of such approximations. An interesting result has shown that the probabilistic approximation given for Dodgson in [CCF$^+$09] has desirable properties that lead the authors to argue that the approximation itself is a reasonable election system. Most notably, the approximation is monotonic, meaning that if a candidate is improved among the preferences of the voters, it will not be given a worst score probabilistically by the approximation algorithm. It is also generally agreed that, despite an equivalent worst-case complexity, the Young election is harder to compute or approximate than the Dodgson election. Another notable result is that a greedy algorithm for Dodgson scoring [HH06] is almost always self-knowingly correct when the voter set is significantly larger than the candidate set. This does not appear to be the case for Young elections either, and it is generally accepted that Young elections are empirically harder to evaluate than Dodgson elections.

2.2 Problems of Interest in Elections

A common problem in all election systems of interest is the incentive for dishonesty. In all election systems of interest, there exist instances in which it is advantageous for some of the voters to vote dishonestly, affecting the outcome of the election to their advantage [Gib73, Sat75, DS00]. Because manipulation is to be carried out by agents with supposedly limited computational power, Bartholdi, Tovey, and Trick [BTT89a] make the observation that manipulation is only a threat to the integrity of the election when the determination of the

manipulation is tractable. It is thus of interest to evaluate the complexity, worst-case or otherwise, of computing how these agents may capitalize on these weaknesses. Two other interesting problems include bribery and control: Elections may be bribed [FHH09], in which an outside agent influences the election by affecting the voters, or controlled [BTT92], in which the chair of the election affects the election by modifying the set of participating voters and candidates.

Each of these problems come in two flavors: constructive, in which our goal is to ensure that a specified distinguished candidate is a winner, and destructive, where our goal is to ensure that such is not a winner. In manipulation, we attempt to reach this goal by giving preferences to a set of unestablished voters, whereas in bribery we do so by changing a given number of votes. There are two subclasses of control problems, those that alter the voter set (i.e., add or delete voters from the election) and those that alter the candidate set.

We define these problems as follows.

Name: $\mathcal{E}$-*Manipulation* [BTT89a, BO91, CSL07]

Instance: A set C of candidates, a set V of established voters, and V' of unestablished voters such that $V \cap V' = \emptyset$, and distinguished candidate p.

Question (constructive manipulation): Does there exist an assignment of preferences for V' such that p is a winner of the $\mathcal{E}$ election with candidate set C and voter set $V \cup V'$?

Question (destructive manipulation): Does there exist an assignment of preferences for V' such that p is not a winner of the $\mathcal{E}$ election with candidate set C and voter set $V \cup V'$?

Another problem of interest is the cases where voters have weights. In such case, we denote the problem by $\mathcal{E}$-weighted-constructive-manipulation and $\mathcal{E}$-weighted-destructive-manipulation.

Name: $\mathcal{E}$-*Bribery* [FHH09]

Instance: A set C of candidates, a set V of voters, distinguished candidate p, and nonnegative integer quota q.

Question (constructive bribery): Is it possible to make p a winner of the $\mathcal{E}$ election by changing the preference profiles of at most q voters in V.

Question (destructive bribery): Is it possible to make p not a winner of the $\mathcal{E}$ election by changing the preference profiles of at most q voters in V.

As in the cases of manipulation, bribery is also defined for cases of weighted voters. As above, we denote this problem $\mathcal{E}$-weighted-bribery. In addition, each voter v can be assigned a price tag $\pi(v)$ of a nonnegative integer. In this case, q is our budget, and we want to achieve our goal (constructive or destructive bribery) by spending at most q in our bribery. We denote this problem by $\mathcal{E}$-\$bribery and $\mathcal{E}$-weighted-\$bribery if voters have both weights and prices. There are six problems of interest in the case of control, as we may add, delete, or partition, either voters or candidates. Also in all of these cases, voters may be weighted.

The cases of constructive control, except the versions used here of adding candidates, are from [BTT92], while those of destructive control, as well as the versions of constructive control by adding candidates used here, are from [HHR07].

Name: *$\mathcal{E}$-Control by Adding Voters*

Instance: A set C of candidates, a set V of established voters, and V' of unestablished voters such that $V \cap V' = \emptyset$, distinguished candidate p, and nonnegative integer quota q.

Question (constructive control): Does there exist a subset $V'' \subseteq V'$ with $||V''|| \leq q$ such that p is a winner of the $\mathcal{E}$ election with candidate set C and voter set $V \cup V''$?

Question (destructive control): Does there exist a subset $V'' \subseteq V'$ with $||V''|| \leq q$ such that p is not a winner of the $\mathcal{E}$ election with candidate set C and voter set $V \cup V''$?

Name: *$\mathcal{E}$-Control by Deleting Voters*

Instance: A set C of candidates, a set V of established voters, distinguished candidate p, and nonnegative integer quota q.

Question (constructive control): Does there exist a subset $V' \subseteq V$ with $||V'|| \leq q$ such that p is a winner of the $\mathcal{E}$ election with candidate set C and voter set $V - V'$?

Question (destructive control): Does there exist a subset $V' \subseteq V$ with $||V'|| \leq q$ such that p is not a winner of the $\mathcal{E}$ election with candidate set C and voter set $V - V'$?

Name: *$\mathcal{E}$-Control by Partitioning Voters*

Instance: A set C of candidates, a set V of established voters, and distinguished candidate p.

Question (constructive control): Does there exist a partition of candidates $V = V_1 \cup V_2$ such that p is a winner of the election with the candidates given by the union of the winners of $E_1 = (C, V_1)$ and $E_2 = (C, V_2)$?

Question (destructive control): Does there exist a partition of candidates $V = V_1 \cup V_2$ such that p is not a winner of the election with the candidates given by the union of the winners of $E_1 = (C, V_1)$ and $E_2 = (C, V_2)$?

Name: *$\mathcal{E}$-Control by Adding Candidates*

Instance: A set C of candidates, a set V of established voters, and C' of unestablished voters such that $C \cap C' = \emptyset$, distinguished candidate p, and nonnegative integer quota q.

Question (constructive control): Does there exist a subset $C'' \subseteq C'$ with $||C''|| \leq q$ such that p is a winner of the $\mathcal{E}$ election with candidate set $C \cup C''$ and voter set V?

Question (destructive control): Does there exist a subset $C'' \subseteq C'$ with $||C''|| \leq q$ such that p is not a winner of the $\mathcal{E}$ election with candidate set $C \cup C''$ and voter set V?

Name: *$\mathcal{E}$-Control by Deleting Candidates*

Instance: A set C of candidates, a set V of established voters, distinguished candidate p, and nonnegative integer quota q.

Question (constructive control): Does there exist a subset $C' \subseteq C$ with $||C'|| \leq q$ such that p is a winner of the $\mathcal{E}$ election with candidate set $C - C'$ and voter set V?

Question (destructive control): Does there exist a subset $C' \subseteq C \backslash \{p\}$ with $||C'|| \leq q$ such that p is not a winner of the $\mathcal{E}$ election with candidate set $C - C'$ and voter set V? In this case, note that we may not deleted the candidate p in which we are manipulating against.

Name: *$\mathcal{E}$-Control by Partitioning Candidates*

Instance: A set C of candidates, a set V of established voters, and distinguished candidate p.

Question (constructive control): Does there exist a partition of candidates $C = C_1 \cup C_2$ such that p is a winner of the election with the candidates given by the union of the winners of $E_1 = (C_1, V)$ and $E_2 = (C_2, V)$?

Question (destructive control): Does there exist a partition of candidates $C = C_1 \cup C_2$ such that p is not a winner of the election with the candidates given by the union of the winners of $E_1 = (C_1, V)$ and $E_2 = (C_2, V)$?

In all of these cases, we say that an election is *computationally vulnerable* to a particular form of misuse, such as manipulation, bribery, or control if the corresponding problem is polynomial-time computable. The election is *computationally resistant* to the misuse if the corresponding problem is NP-hard.

We will also be evaluating the winner problem of election systems in which it is difficult to determine the winner, so-called "hard winner" problems, in particular, Dodgson and Young elections. We define the scores in Dodgson and Young elections as follows.

Definition 2.2.1 *Consider an election $E = (C, V)$ and candidate c. The* Dodgson score *of c is the smallest integer k such that c can be made a Condorcet winner of E by switching k adjacent candidates in the preferences given by V. The* Young score *of c is the smallest integer k such that c can be made a Condorcet winner of E by deleting k voters in V.*

We define decision problems on the score of Dodgson and Young elections as follows.

Name: *Dodgson Score* [BTT89b]

Instance: A Dodgson election $E = (C, V)$, candidate $c \in C$, and nonnegative integer k.

Question: Does c have a Dodgson score of at most k?

Name: *Young Score*

Instance: A Young election $E = (C, V)$, candidate $c \in C$, and nonnegative integer k.

Question: Does c have a Young score of at most k?

Both of these problems are NP-complete [BTT89b].

2.3 Satisfiability and Its Solutions

Satisfiability (SAT) is the problem of determining whether or not a boolean formula can be satisfied by at least one assignment of its variables. SAT was the first known NP-complete problem [Coo71, Lev73], as proven independently by Stephen Cook and Leonid Levin. SAT,

and especially efficient solutions of SAT, is of central importance in many areas of computer science, in which a major challenge is the solving of an NP-complete problem. Even though NP-complete problems are believed to be intractable in the worst case (assuming P $\neq$ NP), this result applies, after all, only in the worst case, and many instances of SAT resulting from practical applications have been shown to be easy to solve by known algorithms.

It is possible to convert a formula into conjunctive normal form (CNF) while only polynomially increasing the number of variables and the formula length [Tse68], while preserving satisfiability. We will assume this representation for the remainder of this section. SAT also remains NP-complete even if all expressions are written in conjunctive normal form with exactly three variables per clause (3-CNF form), resulting in the 3-SAT problem.

Most efficient algorithms for SAT are based on the Davis-Putnam-Logemann-Loveland algorithm (DPLL) [DP60, DLL62], a complete, backtracking-based algorithm. This algorithm operates on formulas in conjunctive normal form. The DPLL algorithm operates by choosing a literal, assigning a truth value, simplifying, and recursively checking if the new formula is satisfiable. If not, the literal is assigned the opposite truth value. As the formula is in CNF, the simplification step consists of removing all clauses which have become true, as well as false literals. If at any branch one or more clauses become empty, the resulting formula is unsatisfiable, and the algorithm backtracks. Satisfiability is determined when all clauses are satisfied. Unsatisfiability can only be detected after an exhaustive search.

There are two pruning techniques for this algorithm. If a clause contains only a single unassigned literal, and all other literals in the clause are assigned false, it can only be satisfied by assigning the necessary value to make this literal true. Also, if a propositional variable occurs with only one polarity in the formula, it can be assigned in a way that makes all clauses containing them true. As an example, consider the formula $x_1 \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (x_2 \vee x_4 \vee x_5) \wedge (\overline{x_1} \vee \overline{x_4} \vee \overline{x_5})$. In this formula, x_1 may be assigned true, as it appears by itself in the first clause, while x_3 may be assigned false, as it appears only in its negation in this formula. Simplifying, we arrive at the formula $(x_2 \vee x_4 \vee x_5) \wedge (\overline{x_4} \vee \overline{x_5})$.

The choice of order for the variables to be assigned, as well as the order of the truth values to assign the literals can make a significant impact on the runtime of the algorithm. Several prominent results in improving this algorithm include variants such as backjumping [Pro93] and clause learning [BKS04].

2.3.1 0-1 Integer Linear Programming and Pseudoboolean Constraints

A related problem to SAT is that of 0-1 Integer Linear Programming (0-1 ILP). 0-1 ILP allows one to express constraints, or clauses, beyond a disjunction of literals. 0-1 ILP is one of Karp's 21 NP-complete problems [Kar72]. This problem can be viewed as a generalization of CNF-SAT, as each disjunctive clause can also be represented as a linear constraint. Reductions, or encodings, to 0-1 ILP have been shown to be valuable in many problems of interest, including Traveling Salesman Problem [MTZ60] and Machine Learning [Wag59].

Name: 0-1 *Integer Linear Programming (*0-1 *ILP)*

Instance: A set of n variables $x_1, \ldots, x_n$ and m constraints $a_{i,1}x_1 + \cdots + a_{i,n}x_n \leq b_i$ for $1 \leq i \leq m$, where $a_{i,j}$ and b_i are integers.

Question: Does there exist an assignment of either 0 or 1 to variables $x_1, \ldots, x_n$ satisfying all m constraints?

In some cases, the problem may also include a linear optimization function which we wish to minimize or maximize. These cases will not be used in the problems of interest in this thesis. For simplicity, we will sometimes write lower-bound and equality constraints in addition to the upper-bound constraints as defined above in our encodings.

In most cases, the problem of interest can be represented by binary variables that are constrained mostly by disjunctive clauses, with a few linear inequalities. In these cases, it is often more reasonable to extend the problem of CNF-SAT, and the algorithms thereof, to accommodate for the few constraints of this form. One natural way to normalize this extension of CNF-SAT is to use pseudoboolean constraints [ES06].

A pseudoboolean constraint has the form $\sum_i w_i \ell_i \geq k$, where each w_i and k is a positive integer and each ℓ_i is a literal. As an example, one may represent the clause $a \vee b \vee \overline{c}$ as $a+b+\overline{c} \geq 1$, and the linear constraint $2a+b-c \geq 1$ can be transformed to $2a+b+(1-c) \geq 2$ or $2a + b + \overline{c} \geq 2$. It should be noted that under this definition, a disjunctive clause is a special case of a pseudoboolean constraint, as the clause $\ell_1 \vee \cdots \vee \ell_m$, where $\ell_1, \ldots, \ell_m$ are literals, can be represented by the constraint $\ell_1 + \cdots + \ell_m \geq 1$. This representation of linear constraints is also interesting as it is both compatible with many algorithms for SAT, and also because these constraints can easily be transformed into disjunctive clauses in other algorithms.

There are many interesting ways to transform pseudoboolean constraints into disjunctive clauses, including the use of adder circuits [Tse68, War98], sorting networks [BB03], and binary decision diagrams [ARMS02a, Min96], with differing tradeoffs, such as consistencies with the initial problem. All of these ideas were applied in the SAT solver MINISAT+ [ES06], and a reduction by adder circuits, proposed by Warners [War98], was used by Connett [Con10] for the reduction of some manipulation problems to SAT.

We will be using solvers for SAT with pseudoboolean constraints to empirically evaluate some problems of interest in election systems, to gain an understanding of the complexity of some of the NP-complete problems of interest. In doing so, we gain an understanding of the differing degrees of hardness of different NP-complete election problems. NP-complete problems occur both in elections of fixed candidate sizes and those with an unbounded candidate size.

Because the conversion of 0-1 Integer Linear Programming instances into SAT with pseudoboolean constraints is fairly straightforward (as seen in the example above), we will be giving reductions to 0-1 ILP in Chapters 5 and 6.

2.4 Partition Problem Variants and Solutions

Another NP-complete problem closely related to manipulation of weighted scoring protocols is Partition. The connection between manipulation of weighted scoring protocols and Partition was first explicitly made by Walsh in the context of manipulating scoring protocols of three candidates [Wal09].

We give the decision version of the Partition problem as follows.

Name: *Partition* [Kar72] (See also [GJ79])

Instance: A multi-set of positive integers $S = \{s_1, \ldots, s_n\}$.

Question: Does there exist a subset $A \subseteq S$ such that $\sum A = \sum (S - A)$?

In the related optimization version of Partition, we wish to minimize the maximum of the two subsets, namely, $\max(\sum A, \sum (S - A))$. Several heuristics and approximation algorithms exist to find relatively good partitions.

We give an example of the connection between election manipulation of three-candidate veto election systems and Partition given in Walsh as follows. Consider a three-candidate

veto election over the candidates p, c_1, and c_2, in which we wish for p to win. Suppose that initially, five voters, of weights 10, 8, 6, 4, and 2 veto p, c_1, p, c_2, and c_2, respectively. In addition, our coalition consists of four voters of weights 6, 5, 4, and 3.

Without loss of generality, none of the four manipulators will veto p. In this example, we must distribute vetoes of weights $\{6, 5, 4, 3\}$ among candidates c_1 and c_2 such that each receives vetoes of total weight at least 16. Currently, c_1 and c_2 have vetoes of total weight 8 and 6. Since $8 - 6 = 2$, this corresponds to a partitioning of the elements $\{6, 5, 4, 3, 2\}$ into two subsets such that each subset has a sum of at least 10. This is possible, as $\{6, 5, 4, 3, 2\} = \{6, 4\} \cup \{5, 3, 2\}$. This corresponds to a manipulation in which the manipulators of weights 6 and 4 vetoing c_2 and those of weights 5 and 3 vetoing c_1.

Because Partition is NP-complete, several heuristics are used to approximate the best partition. Beyond the obvious greedy approach, a heuristic in common use is the Karmarkar-Karp heuristic [KK82]. In the greedy method, one sorts the elements of S in nonascending order and iteratively places each element in the set that minimizes the current difference. In the Karmarkar-Karp heuristic, also known as the differencing heuristic, one decides that the two largest elements are in different sets, deferring deciding in which set each element is placed.

We give an example of the Karmarkar-Karp heuristic as follows. We are given the multi-set $\{6, 4, 3, 3, 2, 2\}$, which we wish to partition. We place 6 and 4 in opposite subsets. By inserting these two elements in opposite subsets, we effectively create a new element equal to the difference of the two largest elements, since we are only concerned about the difference of the sum of the subsets. We create a new element of 2, resulting in multi-set $\{3, 3, 2, 2, 2\}$. We then place each 3 in different subsets, and two elements of 2 in different subsets, resulting in $\{2\}$. In the base case of a single element, we must place it in one of the two subsets. In this case, this algorithm gives a partition of difference 2. Note that in this case the optimal partition has a difference of 0, as $6 + 4 = 3 + 3 + 2 + 2$, and that this heuristic is not always optimal.

Both the greedy and the Karmarkar-Karp heuristic can be extended to complete depth-first tree search algorithms, called Complete Greedy and Complete Karmarkar-Karp algorithms. The Complete Karmarkar-Karp algorithm is given in [Kor98], which we briefly review. We note that in any given instance of Partition, the two largest elements may be either in different subsets or the same subset. By the heuristic, we always try the former first, and terminate if the partition is perfect, or our optimization function is within our desired limit. The search is also pruned if the first element is greater than or equal to the

sum of the remaining elements, as the best partition places the first element in one subset and the remaining elements in the other.

In the pseudocode below, the function **CKK** is given a multi-set of elements $\{x_1, \ldots, x_n\}$, and returns the difference of the sums of the subsets in the optimum partition of the elements into two subsets. The elements are sorted such that $x_1 \geq \cdots \geq x_n$.

If x_1 is at least the sum of the remaining elements, the difference of x_1 and this sum represents the best partition of the set. The remainder of the algorithm finds partitions in which x_1 and x_2 are to go in different or same subsets, returning early if a perfect partition (one with a difference of 0 or 1) is found.

CKK$(\{x_1, x_2, \ldots, x_n\})$ **where** $x_1 \geq \cdots \geq x_n$

if $(x_1 \geq \sum_{2 \leq i \leq n} x_i)$
    return $x_1 - \sum_{2 \leq i \leq n} x_i$

$\texttt{diff}_1 \leftarrow$ **CKK**$(\{x_1 - x_2, x_3, \ldots, x_n\})$
if $(\texttt{diff}_1 \leq 1)$ **return** $\texttt{diff}_1$

$\texttt{diff}_2 \leftarrow$ **CKK**$(\{x_1 + x_2, x_3, \ldots, x_n\})$
return $\min(\texttt{diff}_1, \texttt{diff}_2)$

Partition has been extended to the problem of k-Way Partition, in which the goal is to divide the set into k equal subsets. Korf notes in [Kor10] that there are at least three optimization functions of interest in k-Way Partition: we may wish to minimize the maximum subset sum, maximize the minimum subset sum, or minimize the maximum difference between the sums of each two subsets. In the case of 2-Way Partition, these optimizations are identical. Korf [Kor10] showed that all three of these optimization functions can produce different optimal partitions for $k \geq 3$. The first two functions are of interest in some manipulation problems we will evaluate in this thesis. The first optimization is interesting because we may want the maximum score given to the nondistinguished candidates not to exceed the final score of our distinguished candidate in a manipulation problem. The second optimization is of interest in cases where vetoes are counted. We define the problem and these two optimizations as follows.

Name: *k-Way Partition* [Kar72] (See also [GJ79])

Instance: A multi-set of positive integers $S = \{s_1, \ldots, s_n\}$.

Question (decision): Are there disjoint and covering subsets $S = A_1 \cup \cdots \cup A_k$ such that $\sum A_1 = \cdots = \sum A_k$?

Question (optimization #1): Find disjoint and covering subsets $S = A_1 \cup \cdots \cup A_k$ such that $\max(\sum A_1, \ldots, \sum A_k)$ is minimized.

Question (optimization #2): Find disjoint and covering subsets $S = A_1 \cup \cdots \cup A_k$ such that $\min(\sum A_1, \ldots, \sum A_k)$ is maximized.

An early result by Korf [Kor98] showed how the greedy and Karmarkar-Karp heuristics, as well as the corresponding complete algorithms, can be extended from Partition to k-Way Partition. In the case of the Complete Greedy Algorithm, we search a k-ary tree in which we try inserting elements, in nonascending order, into each of the k subsets, trying the element in each subset in nondescending order by the current sum. The Karmarkar-Karp heuristic works by tracking partial partitions. For example, in a 3-Way Partition of the set $\{4, 5, 6, 7, 8\}$, our initial subpartitions are $\{(8, 0, 0), (7, 0, 0), (6, 0, 0), (5, 0, 0), (4, 0, 0)\}$, sorted in nonascending order by the largest element in the partition. The two subpartitions containing the largest elements are combined into a single subpartition greedily, to minimize the largest sum, in this case, resulting in the set of tuples $\{(8, 7, 0), (6, 0, 0), (5, 0, 0), (4, 0, 0)\}$. The next step results in the tuple $(8, 7, 6)$. Since only the difference within tuples is of importance, we may normalize this tuple to $(2, 1, 0)$. In the corresponding complete CKK algorithm, we will need to try each combination of the largest two tuples, of which there may be as many as $k!$, in nondecreasing order of the greatest sum produced. In the example above, suppose we wish to combine the tuples $(8, 7, 0)$ and $(6, 0, 0)$. There are three distinct combinations in this case: $(8, 7, 0) + (0, 0, 6) = (8, 7, 6)$, $(8, 7, 0) + (0, 6, 0) = (13, 8, 0)$, and $(8, 7, 0) + (6, 0, 0) = (14, 7, 0)$. We try these in nondecreasing order of the largest element, in this case, in the order $(8, 7, 6)$, $(13, 8, 0)$, and $(14, 7, 0)$. Note that there are $k!$ ways that two k-element tuples may be combined in the case of k-Way Partition. The search can be pruned if the largest element of the first tuple cannot be combined with the smallest element of the remaining tuples to give a better partition. This is a generalization of the pruning used in CKK of Partition.

In [Kor09], the empirical complexity of the Complete Greedy Algorithm (CGA) and the Complete Karmarkar-Karp (CKK) Algorithm for k-Way Partition are evaluated. Interestingly, while CKK was found to be more efficient empirically than the CGA for 3-Way

Partition, the opposite was found for k-Way Partition for $k \geq 4$. This is likely due to the complexity of combining the running totals of the sets as there are $k!$ combinations for every two tuple of k sums, whereas the greedy algorithm has at most k branches, one for each subset to place the element, at each level.

Two new algorithms for k-Way Partition, Sequential Number Partitioning (SNP) and Recursive Number Partitioning (RNP) were also introduced in [Kor09]. Both algorithms operate recursively on k, the number of subsets. In Sequential Number Partitioning, one complete subset is first chosen, and the remaining unpartitioned numbers are partitioned recursively using this algorithm for $(k-1)$-Way Partition. In the base case, 2-Way Partition is evaluated using the CKK algorithm. The choice of the first subset is enumerated using an inclusion-exclusion tree search; at each search node, we are to decide whether or not to include a particular element.

There are several pruning techniques for the search of this first complete subset. First, to avoid symmetry, the sum of the first subset is restricted to be no more than $\lfloor \frac{t}{k} \rfloor$, where t is the sum of the elements, and the subsets are subsequently chosen in nondescending order by sum. Also, if m is the maximum subset sum of the best partition found thus far, or our target maximum, we restrict the sum of the first subset to be at least $t - (k-1)m$, as otherwise the best partition found utilizing this first subset sum cannot improve upon this difference. To induce as much pruning as early as possible in the search tree, the elements are considered in nonascending order, and we try inclusion in our first branch and exclusion second.

In the pseudocode below, we are partitioning the set $S = \{s_1, \ldots, s_n\}$ into k subsets. The parameter `min` tells us the minimum that the first subset should sum to. This is $\lfloor \frac{t}{k} \rfloor$ for the first subset, and will be updated with the sum of the previous subset, to ensure nondescending order of the subsets, breaking symmetry. The variable `best` tracks the best partition found so far. In the code below, we are minimizing the sum of the largest subset by sum, returning the smallest possible largest subset sum. This variable allows us to prune by disregarding branches in which it is impossible to find a better partition. The variable $1 \leq i \leq n$ allows us to traverse the elements of s, deciding which elements to include or exclude. The sum of the elements we have included thus far is stored in the parameter `sum`, and the subset of elements excluded in S'. The elements of S' will be recursively partitioned by $k-1$ ways.

In the first conditional of $k = 1$, or 1-Way Partition, we are checking for the base case condition, in which we trivially return the sum of the elements. The second conditional of

$i > n$ tells us that we have evaluated each element of S for inclusion into the first subset, and need to recursively call the algorithm for $(k-1)$-Way Partition. In the $(k-1)$-Way Partition instance, the second subset must sum to at least that of the first subset, `sum`. This is accounted for by the third parameter given to **SNP**. The third and fourth conditional in the function below account for including or excluding the element s_i, respectively.

SNP($S = \{s_1, \ldots, s_n\}$,k, **where** $x_1 \geq \cdots \geq x_n$
`min`, `best`,
i, `sum`, S')

if $(k = 1)$
return $\sum_{1 \leq j \leq n} s_j$

if $(i > n)$
return **SNP**$(S', k-1, \texttt{sum}, \texttt{best}, 0, 0, \emptyset)$

if $(\texttt{sum} + s_i \leq \texttt{best})$
$\texttt{best}_1 \leftarrow$ **SNP**$(S, k, \texttt{min}, \texttt{best}, i+1, \texttt{sum} + s_i, S')$
$\texttt{best} \leftarrow \min(\texttt{best}, \texttt{best}_1)$

if $(\texttt{sum} + \sum_{i+1 \leq j \leq n} s_j \geq \texttt{min})$
$\texttt{best}_2 \leftarrow$ **SNP**$(S, k, \texttt{min}, \texttt{best}, i+1, \texttt{sum}, S' \cup \{(w_i, \ell_i)\})$
$\texttt{best} \leftarrow \min(\texttt{best}, \texttt{best}_2)$

return `best`

In Recursive Number Partitioning, on an instance of k-Way Partition for k even, the set is first divided into two subsets, each of which will be partitioned $\frac{k}{2}$ ways, by a top-level CKK search. If k is odd, an iteration of SNP is used initially.

In the pseudocode below, the function **RNP** is given a superset of integers, each one, x_i labeled with subsets A_i, B_i such that $x_i = \sum_{w \in A_i} w - \sum_{w \in B_i} w$. It is assumed that the elements are sorted in nonasscending order $x_1 \geq \cdots \geq x_n$. The elements in A_i and B_i tell us the initial elements contributing to the difference of x_i. Recall that we wish to minimize the

maximum subset, returning the sum of the smallest possible maximum subset.

In the base-case in which $n = 1$, we have enumerated one possible 2-way partition of the elements. We recursively partition the elements of A_1 and B_1, each side of the 2-way partition, into $\frac{k}{2}$ subsets, finding the best partition of each. The maximum of the two results represents the value of the overall k-way partition.

Given the two largest elements, $x_1 = \sum_{w \in A_1} w - \sum_{w \in B_1} w$ and $x_2 = \sum_{w \in A_2} w - \sum_{w \in B_2} w$, a top-level CKK algorithm can combine them by difference or sum, joining the respective subsets accordingly in each case. To ensure it is still possible to find a partition beating the best found thus far, we check that the elements of each of the new subsets generated sum to at most $\frac{k}{2}(\texttt{best} - 1)$. The best of the two cases is returned as the best partition.

RNP($S = \{x_1 = \sum_{w \in A_1} w - \sum_{w \in B_1} w,$
$x_2 = \sum_{w \in A_2} w - \sum_{w \in B_2} w, \ldots,$
$x_n = \sum_{w \in A_n} w - \sum_{w \in B_n} w\}$,
k,
$\min, \texttt{best}$)

if $(n = 1)$; Base case, we are done with the top-level CKK division
$\texttt{best}_1 \leftarrow \texttt{RNP}(A_1, \frac{k}{2}, \min, \texttt{best})$; Recursively partition A_1 $\frac{k}{2}$ ways.
if $(\texttt{best}_1 \geq \texttt{best})$ **return** best. ; Can't beat the best partition found.
$\texttt{best}_2 \leftarrow \texttt{RNP}(B_1, \frac{k}{2}, \min, \texttt{best})$; Recursively partition B_1 $\frac{k}{2}$ ways.
if $(\texttt{best}_2 \geq \texttt{best})$ **return** best. ; Did not beat the best partition.
return $\max(\texttt{best}_1, \texttt{best}_2)$; Return overall result of this partition

else
if each of $A_1 \cup B_2$ and $B_1 \cup A_2$ sum to at most $k(\texttt{best} - 1)$
$\texttt{best}_1 \leftarrow \texttt{RNP}(\{(x_1 - x_2) = \sum_{w \in A_1 \cup B_2} w - \sum_{w \in A_2 \cup B_1} w, \ldots,$
$x_n = \sum_{w \in A_n} w - \sum_{w \in B_n} w\}$,
$k, \min, \texttt{best}$)
if $(\texttt{best}_1 < \texttt{best})\texttt{best} \leftarrow \texttt{best}_1$; Update best partition found

if each of $A_1 \cup A_2$ and $B_1 \cup B_2$ sum to at most $k(\texttt{best} - 1)$

$\texttt{best}_2 \leftarrow \texttt{RNP}(\{(x_1 + x_2) = \sum_{w \in A_1 \cup B_1} w - \sum_{w \in A_2 \cup B_2} w, \ldots,$
$x_n = \sum_{w \in A_n} w - \sum_{w \in B_n} w\},$
$k, \min, \texttt{best})$

if $(\texttt{best}_2 < \texttt{best})\texttt{best} \leftarrow \texttt{best}_2$

return best

As an example, consider the problem of 4-Way Partition on the same five-element subset of $\{8, 7, 6, 5, 4\}$. The first branch of the CKK algorithm is merely that of the Karp-Karmarkar heuristic, and proceeds $\{6, 5, 4, 1 = 8-7\}$, $\{4, 1 = 8-7, 1 = 6-5\}$, $\{3 = (7+4)-8, 1 = 6-5\}$, finally yielding the partition of $\{7, 5, 4\} \cup \{8, 6\}$. A recursive call to the CKK algorithm on each of these subsets produces a partition in which the sum of the largest subset is 9. By backtracking, we see that it is also possible to combine the elements $3 = (7 + 4) - 8$ and $1 = 6-5$ by sum, yielding the 2-way partition of $\{7, 6, 4\} \cup \{8, 5\}$. However, as $7+6+4 = 17$ is more than 2(8), this cannot yield a better overall partition, and we need not further evaluate this branch.

In [Kor09], it is shown that both the SNP and RNP algorithms show a marked improvement over CKK for k-Way Partition when $k \geq 3$, with RNP significantly faster than SNP for $k \geq 4$. One particular unfortunate issue is that RNP cannot easily be extended for k odd, and iterations of the less-efficient SNP algorithm are required for these cases. Most notably, 3-Way Partition is almost as complex as 4-Way Partition in many cases, and 5-Way Partition is significantly more complex than either.

Although it is mentioned in [Wal09] that manipulation of scoring protocols of three candidates, as well as veto elections of more than three candidates can be solved using the solutions of multi-way partition, some adjustments must be made to this problem. We give an example and demonstrate the adjustments as follows.

Consider a four-candidate veto election over the candidates p, c_1, c_2, and c_3, in which we wish for p to win. Suppose that initially, four voters, of weights 20, 12, 9, and 7 veto p, c_1, c_2, and c_3 respectively. In addition, our coalition consists of six voters of weights 10, 8, 4, 4, 3, and 3.

In this example, we must distribute vetoes of weights $\{10, 8, 4, 4, 3, 3\}$ among candidates c_1, c_2, and c_3, currently with vetoes of total weight 12, 9, and 7, such that each receives

vetoes of weight totaling at least 20.

Since $12-9=3$ and $12-7=5$, we are interested in partitions of the set $\{10,8,4,4,3,3\}\cup\{3,5\}$. However, in this case, we are interested in partitions in which 3 and 5 are not placed in the same subset, as these two subsets represent the vetoes given to candidates c_2 and c_3, respectively. This requires some adjustments to the partition problem and algorithms of interest, which our work in later sections encompasses.

Furthermore, as k-Way Partition problems partition only individual numbers, the application of this problem and its algorithms to that of manipulation of scoring protocols is limited to cases of plurality and veto, in which each vote is defined by a single candidate. For general scoring protocols such as Borda, we will need to introduce analogous partition problems.

In our new problem, k-Way Permutation Partition for $k \geq 2$, we are given a multiset of tuples, each of cardinality k. We wish to find permutations of each tuple such that, if we take the sum of each of the k positions among the tuples, the k sums are equal. As in the case of k-Way Partition, there are a few interesting optimization functions, of which two are of interest to manipulation problems. We give a formal definition of what a permutation partition is as follows.

Definition 2.4.1 *Let $S=\{x_1,\ldots,x_n\}$ be a multi-set of k-tuples, where $x_i=(x_i^1,\ldots,x_i^k)$. A* k-way permutation partition *of S is a mapping $P:\{1,\ldots,n\}\rightarrow {S_{\{1,\ldots,k\}}}^1$, assigning a permutation over the k elements to each of the n tuples. The i^{th}* subset *of this permutation partition is the set of elements $\{x_1^{P(1)_i},\ldots,x_n^{P(n)_i}\}$.*

The decision and optimization problems of *k-Way Permutation Partition* attempts to find a k-way permutation partition that minimizes disparities in the sum of the k subsets. We define these problems as follows.

Name: *k-Way Permutation Partition*

Instance: A multi-set of k-tuples, $S=\{x_1,\ldots,x_n\}$, where $x_i=(x_i^1,\ldots,x_i^k)$. It can be assumed without loss of generality that $x_i^k=0$ for all $1\leq i\leq n$, as one may normalize the tuple, noting that only the difference among tuple elements is crucial.

Question (decision): Does there exist a mapping $P:\{1,\ldots,n\}\rightarrow S_{\{1,\ldots,k\}}$ such that

$$\sum_{1\leq r\leq n} x_r^{P(r)_1} = \cdots = \sum_{1\leq r\leq n} x_r^{P(r)_k}?$$

[1] S_A is the set of all permutations over A.

Question (optimization #1): Find the mapping $P : \{1, \ldots, n\} \rightarrow S_{\{1,\ldots,k\}}$ that minimizes

$$\max_{1 \leq i \leq k} \sum_{1 \leq r \leq n} x_r^{P(r)_i}.$$

Question (optimization #2): Find the mapping $P : \{1, \ldots, n\} \rightarrow S_{\{1,\ldots,k\}}$ that maximizes

$$\min_{1 \leq i \leq k} \sum_{1 \leq r \leq n} x_r^{P(r)_i}.$$

In this problem, intuitively, we are "partitioning" the set S into the $k!$ possible permutations of each k-tuple such that the sum of the elements in each of the k positions is reasonably uniform. By a "partition," we are referring to a permutation of the set of tuples of S satisfying the constraints described above.

Note that k-Way Partition is a special case of this problem, in which each k-tuple has the form $(x_i^1, 0, \ldots, 0)$. We also note that because it is computationally easy to check the validity of such a partition, this problem is NP-complete.

In the example above, initially, the candidates c_1, c_2, and c_3 received vetoes initially of total weight 12, 9, and 7, respectively. There were six manipulators of weight 10, 8, 4, 4, 3, and 3. Because we assume without loss of generality that none of these manipulators will veto p, they will give vetoes to c_1, c_2, or c_3. As p must defeat or tie each of these three candidates, each of these candidates must receive vetoes totaling at least 20 following manipulation. This means we must partition the tuples $\{(12, 9, 7), (10, 0, 0), (8, 0, 0), (4, 0, 0), (4, 0, 0), (3, 0, 0), (3, 0, 0)\}$ such that each subset has a sum of at least 20. This is similar to the approach taken by the CKK algorithm for Multi-Way Partition. We may simplify the first tuple by subtracting 7 from each element, making $(5, 2, 0)$, and modify our target sum to $20 - 7 = 13$.

As we will see in a Chapter 4, it is sometimes more efficient to count the total number of approvals given by the voters, while in other cases such as above, to count the total number of vetoes. We give an example involving the Borda count election below.

Suppose we have four candidates p, c_1, c_2, and c_3, and that initially, four voters of weights 20, 12, 9, and 7 have preferences $c_3 \succ c_2 \succ c_1 \succ p$, $p \succ c_1 \succ c_2 \succ c_3$, $c_1 \succ p \succ c_3 \succ c_2$, and $c_2 \succ c_1 \succ p \succ c_3$, respectively. Initially, p, c_1, c_2, and c_3 thus have scores 61, 85, 73, and 69 under the Borda count. As in the earlier example, we have six manipulators, of weights 10, 8, 4, 4, 3, and 3. Without loss of generality, each manipulator will choose p as his or her first preference. This will give p a final score of 157.

The manipulator of weight 10 will give scores of 20, 10, and 0 to candidates c_1, c_2, and c_3. Each of c_1, c_2, and c_3 must finish the election with a total score of at most 157. Thus, the tuples $\{(85, 73, 69), (20, 10, 0), (16, 8, 0), (8, 4, 0), (8, 4, 0), (6, 3, 0), (6, 3, 0)\}$ must be partitioned such that each subset has a total sum of at most 157.

In Chapter 4, we will formulate extensions of algorithms introduced by Korf in [Kor98, Kor09] for Partition and k-Way Partition to k-Way Permutation Partition. We also evaluate the relationship of this problem to that of manipulating some scoring systems, the distribution of the complexity of instances in evaluating such using these algorithms, and thus the frequency of hard instances of this problem in some systems.

We confirm the statement of Walsh that the result that hardness in manipulation does not follow from the phase transition of the problem, also hold for veto elections of four or more candidates. The result stems from the same empirical data: that we can find a manipulation or show that none exist in veto elections in very few search nodes using the depth-first search Partition algorithms given by Korf, and generalizations thereof that we will introduce in Chapter 4.

Our new computational problem of k-Way Permutation Partition also allow one to evaluate manipulation in the general case of k-candidate scoring protocols. We investigate the complexity of manipulating scoring protocols using new algorithms we derive by extending the known algorithms for Multi-Way Partition.

2.5 Edge Covering and Edge Matching Problems

We will show in this thesis that some problems in some election systems are closely related to problems involving edge coverings. This is especially true for elections that distinguish two candidates from the remaining candidates. Due to this connection, these problems generally have similar complexities to that of some forms of edge covering problems. An edge cover, as well as the well-known decision problem of Edge Cover, is defined as follows.

Definition 2.5.1 *An* edge cover *of a graph is a set of edges such that every vertex of the graph is incident to at least one edge of the set.*

Name: *Edge Cover* [Kar72] (See also [GJ79, pages 79, 190])

Instance: An undirected graph $G = (V, E)$ and positive integer q.

Question: Does there exist an edge cover $C \subseteq E$ for G of size at most q?

It is possible to find a smallest edge cover in polynomial time, by finding a maximum matching and extending it greedily so that all vertices are covered. We will use several interesting variations of Edge Cover in this thesis, which are defined as follows.

In the variation b-Edge Cover, each vertex v is to be covered by a minimum of some number, $b(v)$, of edges. There are several interesting variations of this problem: Each edge can be chosen only once (Simple b-Edge Cover), an arbitrary number of times (b-Edge Cover), or have a capacity and be chosen up to that many times (Capacitated b-Edge Cover) (see [Sch03] sections 34.1, 34.7, and 34.8), all of which are polynomial-time computable [Pul73, CI78, Gab83, Ans87]. In each case, polynomial-time computability is due to the relationship between edge coverings and edge matchings. More specifically, the size of the minimum edge covering and the size of the maximum edge matching always sum to the total of the b-values. The maximum edge matching can then be computed using linear programming.

We further extend this problem to that of multigraphs, defining Simple b-Edge Cover of Multigraphs as follows.

Name: *Simple b-Edge Cover of Multigraphs*

Instance: An undirected multigraph $G = (V, E)$, a function $b : V \to \mathbf{Z}^+$ and positive integer q.

Question: Does there exist a subset of at most q edges $C \subseteq E$ such that each vertex $v \in V$ is incident to at least $b(v)$ edges in C?

This variation is also polynomial-time computable, as follows.

Theorem 2.5.2 *Simple b-Edge Cover for Multigraphs is polynomial-time computable.*

Proof: We demonstrate how to compute Simple b-Edge Cover of Multigraphs using Capacitated b-Edge Cover, which we define as follows.

Name: *Capacitated b-Edge Cover* (See also [Sch03, chapter 34])

Instance: An undirected graph $G = (V, E)$, a function $b : V \to \mathbf{Z}^+$, a capacity function $c : E \to \mathbf{Z}^+$, and positive integer q.

Question: Does there exist a multisubset of edges $C \subseteq E$ such that each vertex $v \in V$ is incident to at least $b(v)$ edges in C and such that each $e \in E$ is chosen by C with frequency at most $c(e)$?

Let $G = (V, E)$, $b : V \to \mathbf{Z}^+$, and $q > 0$ be an instance of Simple b-Edge Cover of Multigraphs.

We construct an instance of Capacitated b-Edge Cover as follows. Let G' be a complete graph of V. For each pair $v_1, v_2 \in V$, we set the capacity $c(v_1, v_2)$ to the number of edges between v_1 and v_2 in G. ■

Simple b-Edge Cover of Multigraphs, and several variations thereof, is of interest in solving some election manipulation problems in which the voters always distinguish exactly two candidates from the remaining candidates. More specifically, in such a construction, the vertices correspond to candidates and the edges to the voters, who distinguish two candidates from the remaining. We introduce two additional versions of Simple b-Edge Cover of Multigraphs, which will be of interest to problems involving weights and prices for voters.

Name: *Simple Weighted b-Edge Cover of Multigraphs*

Instance: An undirected multigraph $G = (V, E)$, a function $b : V \to \mathbf{Z}^+$, weight function $w : E \to \mathbf{Z}^+$ and positive integer q.

Question: Does there exist a subset of edges $C \subseteq E$ of total weight at most q such that each vertex $v \in V$ is incident to at least $b(v)$ edges in C?

Name: *Simple b-Edge Weighted Cover of Multigraphs*

Instance: An undirected multigraph $G = (V, E)$, a function $b : V \to \mathbf{Z}^+$, weight function $w : E \to \mathbf{Z}^+$ and positive integer q.

Question: Does there exist a subset of at most q edges $C \subseteq E$ such that each vertex $v \in V$ is incident to edges in C of at least total weight $b(v)$?

In these two versions, as we will see later, the weighted edges correspond to cases of bribery in which each voter has an associated price. The weighted coverings correspond to manipulations involving weighted voters. It is currently unknown whether either of these variations are polynomial-time computable, NP-intermediate, or NP-complete. We will show in this thesis that some problems of interest are polynomial-time equivalent to one of these problems.

A related problem involves edge matchings. See [Sch03] for the connection between matchings and coverings, and the extension of Edge Matching to b-Edge Matching, and the

polynomial-time computability result. We will define an analogous problem for multigraphs as follows.

Name: *Simple b-Edge Matching of Multigraphs*

Instance: An undirected multigraph $G = (V, E)$, a function $b : V \to \mathbf{Z}^+$ and positive integer q.

Question: Does there exist a subset of at least q edges $C \subseteq E$ such that each vertex $v \in V$ is incident to at most $b(v)$ edges in C?

This is an extension of Edge Matching. In a traditional Edge Matching problem, each vertex has a b-value of $b(v) = 1$, as a matching cannot cover a vertex more than once.

2.6 Other Related Problems

In addition to SAT and 0-1 ILP, we will be using some other problems outside of computational social choice theory, both in solving problems in election systems and also in proving the difficulty of doing so. In particular, the NP-complete problems of Exact Cover by 3-Sets (X3C) and Hitting Set are well-suited to show hardness of some problems of interest, particularly manipulation in unweighted elections.

Exact Cover by 3-Sets (X3C) is a common variation of a restricted version of Set Cover.

Name: *Exact Cover by 3-Sets (X3C)* [Kar72] (See also [GJ79, problem SP2])

Instance: A set $S = \{s_1, \ldots, s_{3m}\}$ and subsets $T_1, \ldots, T_n \subseteq S$ such that $||T_i|| = 3$.

Question: Does there exist a set $A \subseteq \{1, \ldots, n\}$ such that $||A|| = m$ and $\bigcup_{i \in A} T_i = S$?

Other variations of restricted set cover type problems also exist, such as Exact $\frac{3}{4}$-Set Cover [FHS08], which was used to show the NP-completeness of bribery in the Borda count election [BFH$^+$08].

An important problem used in showing complexities of problems involving control by adding or deleting candidates is Hitting Set.

Name: *Hitting Set* [Kar72] (See also [GJ79, problem SP8])

Instance: A set $S = \{s_1, \ldots, s_m\}$, n subsets of S, $T_1, \ldots, T_n$, and positive integer $1 \leq q \leq m$.

Question: Does there exist a subset of q elements of S, $S' = \{s_{i_1}, \ldots, s_{i_q}\}$, such that $S' \cap T_i \neq \emptyset$ for each $1 \leq i \leq n$? Thus, each subset T_i contains at least one element of S'.

Hitting Set can also be viewed as a generalization of Vertex Cover to hypergraphs, where edges connect an arbitrary number of vertices. This problem was used to prove that unweighted plurality elections are computationally resistant to control by adding and deleting candidates in [BTT92]. We will generalize this result.

2.7 Succinct Preference Representation

An obvious way to represent the preference profile of an election is to list the preferences of each voter. However, in an unweighted election, the effects of different voters with identical preferences are identical, and as such, it is also possible to give the preference profile by listing the distinct preference orderings of the election and counting the number of voters with each distinct preference. For example, in an unweighted election of three candidates a, b, and c, there are $3! = 6$ distinct preferences that may be given by the voters. The preference profile of the election may be given by indicating how many voters prefer each of the six distinct preferences (see also [FHH09]).

An important property of succinct preference representation is that for a fixed candidate set size m, there is a bounded, at most $m!$, number of distinct preference orderings. This enables one to solve some problems that are difficult with large voter sets in polynomial time, due to the constant bound on the problem size. A particularly important tool for exploiting this property is Integer Linear Programming (ILP), including the 0-1 ILP variant. ILP is polynomial-time computable for instances of a bounded number of variables [Len83]. Using succinct preferences, it has been found that manipulating scoring protocols [FHH09], finding the Dodgson score of a fixed number of candidates [BTT89b], and some other problems which are NP-hard in the general case, are polynomial-time computable when the candidate size is bounded. However, it should be noted that the degree of the polynomial upper-bound of the runtime may be significant for moderate-sized candidate sets, due to the exponential nature of $m!$.

Succinct preference profiles are also useful for breaking symmetry in the evaluation of manipulation and other NP-hard problems. For example, if we wish to find a deletion of unweighted voters to make c, the distinguished candidate, a winner, a linear listing of voters

may encourage an algorithm to attempt many similar deletions in its search, as deleting voters with identical preference orderings is not distinguishable for the purposes of this manipulation. A succinct representation, on the other hand, allows one to represent the cardinality of voters with each distinct preference ordering to be affected by the manipulation. We will be using succinct preference profiles for symmetry breaking purposes in manipulation algorithms in Chapter 5 and for evaluating Dodgson and Young elections in Chapter 6.

2.8 Bounding Dodgson and Young Scores

Despite the NP-hardness of scoring candidates in Dodgson and Young elections, polynomial-time algorithms exist for approximating and bounding Dodgson and Young scores. In Chapter 6, we will make use of such bounds for computing scores and winners in both cases. In each of the results below, polynomial-time computability also holds for succinct input.

A greedy approximation algorithm for approximating the Dodgson score of a candidate c in an election is given by [HH06]. In [CCF$^+$09], a randomized approximation algorithm is given, and its properties are contrasted with those of the deterministic greedy algorithm in [HH06]. Because the deterministic greedy algorithm finds a series of swaps making c a Condorcet winner in the election, it finds an upper bound for the Dodgson score of c. We review it briefly and show how the same principle can be applied to finding upper bounds of the Young scores. We also show lower bounds for the Dodgson and Young scores.

Consider an election $E = (C, V)$. Let c and c' be candidates in C. Define the deficit of c against c', $\texttt{def}(c, c')$, as the minimum number of voters in which c must be pushed ahead of c' in order for c to have a majority over c' among the voters of V. For example, if currently six voters prefer c over c' and ten vice versa, then the deficit of c against c', or $\texttt{def}(c, c')$ is three. c must be pushed above c' in the preference profiles of at least three voters who currently prefer c' to c. The total deficit of c is the sum of the deficits of c against each other candidate.

Without loss of generality, to make c a winner by swapping adjacent candidates in the preferences of the voters, we will only make swaps that push c up in the preferences of voters. c may be pushed up some number of positions within each voter's preferences in the change. In the greedy algorithm for Dodgson score, we iteratively make the most cost effective push. The cost effectiveness of a push is defined by the ratio that the total deficit of c is reduced to the number of positions c is moved upward. We iteratively perform the most

cost effective push until c is a Condorcet winner. This algorithm clearly gives an upper bound for the Dodgson score of c, and it is shown in [CCF+09] that this algorithm approximates the Dodgson score by a factor of H_{m-1}, where $m = ||C||$ is the number of candidates and $H_i = \sum_{1 \leq j \leq i} \frac{1}{j}$ is the i^{th} harmonic number.

A lower bound of the Dodgson score of c can be given by the total deficit of c in the election, as clearly, each adjacent swap can reduce the total deficit of c by at most one.

We define a very similar greedy algorithm for the Young score as follows. The algorithm will make use of a different definition of the deficit. The deficit of c against c', $\texttt{def}(c, c')$ will be defined as the minimum number of voters in which c' is outperforming c that must be removed such that c beats c' in a pairwise election. We define the maximum deficit of c to be the maximum of the deficit of c against any other candidate, or $\max_{c' \neq c} \texttt{def}(c, c')$.

In our greedy algorithm for computing the Young score of c, we iteratively delete the voter which reduces the maximum deficit of c the most, until c becomes a Condorcet winner. This gives us an upper bound for the Young score of c. The maximum deficit of c itself is a lower bound for the Young score of c. This is because removing one voter from the election can reduce the deficit of c against any candidate c' by at most one, and thus, reduce the maximum deficit of c by at most one.

In the pseudocode below for finding the greedy approximation to the Young score, we are given an election $E = (C, V)$ and a candidate $c \in C$ for which we wish to score. We check if c is a Condorcet winner or if the voter set is empty, and return a score of 0 if this is the case. If not, we check to see if deleting any of the voters will reduce the maximum deficit of c, $\max_{c' \neq c} \texttt{def}(c, c')$, by one. If such a voter exists, we choose one such voter and delete it. If not, it suffices to delete any voter.

GreedyYoungScore$(E = (C, V), c \in C)$

if $(\max_{c' \neq c} \texttt{def}_E(c, c') = 0)$
  return 0

if $(V = \emptyset)$
  return 0

for each $v \in V$
  $E' \leftarrow (C, V \backslash \{v\})$

if $(\max_{c' \neq c} \mathtt{def}_{E'}(c, c') = \max_{c' \neq c} \mathtt{def}_{E}(c, c') - 1)$
return $1 +$ **GreedyYoungScore**(E', c)

return $1 +$ **GreedyYoungScore**$((C, V \backslash \{v_0\}), c)$ for any $v_0 \in V$

We will be using these lower and upper bounds of Dodgson and Young scores, all of which are polynomial-time computable, for pruning purposes in finding Dodgson and Young winners in Chapter 6.

2.9 Phase Transitions of NP-Complete Problems

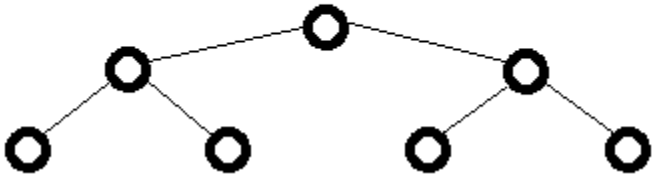
It is well-known that most instances of many NP-complete problem, such as SAT, are easy to solve, and that computationally hard cases are rare. One model of understanding where the hard problems are is the phase transition.

Almost all NP-complete problems exhibit a phase transition, in which the probability (with respect to a uniform distribution) of an accepting instance abruptly changes from 0 to 1 when a constrainedness parameter, also called an order parameter, is adjusted [TCK91]. Intuitively, the problem goes from being underconstrained, with a high density of solutions, to being overconstrained, and very unlikely to contain a solution, as a certain threshold of constrainedness is passed. The phase transition has been observed to be a significant challenge to many algorithms of interest, in which the complexity often peaks at the critical value for the parameter. In this region, the probability of a solution is low but nonnegligible. For example, it is known that in the problem of SAT for 3-CNF formulas, phase transition occurs when the constrainedness parameter of $\frac{n}{m}$, where n is the number of clauses and m the number of variables approaches a critical value of about 4.2 [GW96]. This occurs because with fewer than $4.2m$ clauses, the variables are likely underconstrained and it is easy to find a solution, since there are a lot of solutions. On the other hand, with more than $4.2m$ clauses, the algorithm is likely to backtrack very early, because a contradiction is reached very easily. At the boundary of roughly $4.2m$ clauses, there is often a high density of well-separated almost solutions which causes most backtracking algorithms to search very deeply in the search tree, or "thrash" [TCK91]. It is further conjectured that all NP-complete problems have at least one order parameter in which hard-to-solve problems are around a critical value of this parameter. The probability of a solution abruptly changes from almost zero to almost

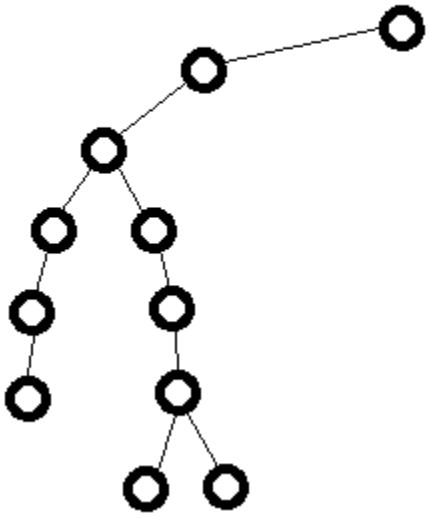
one at this transition.

In the diagrams below, we show examples of search trees in problem instances that are overconstrained or underconstrained, respectively.

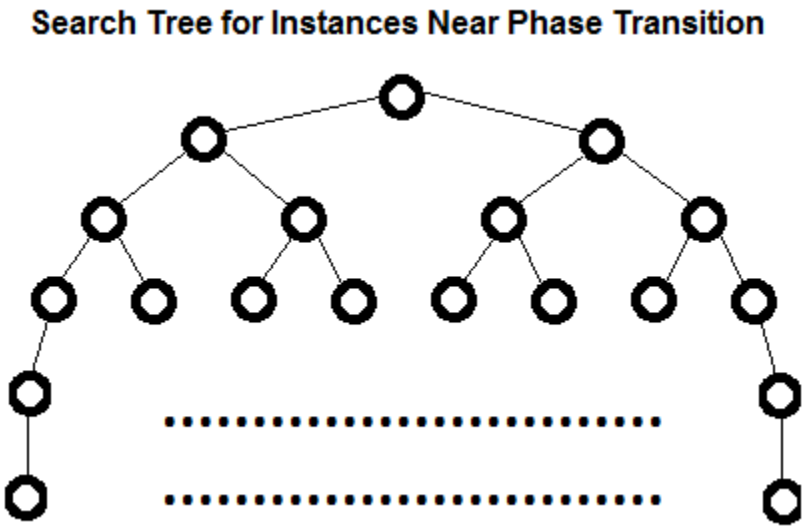
Search Tree for Overconstrained Cases



Search Tree for Underconstrained Cases

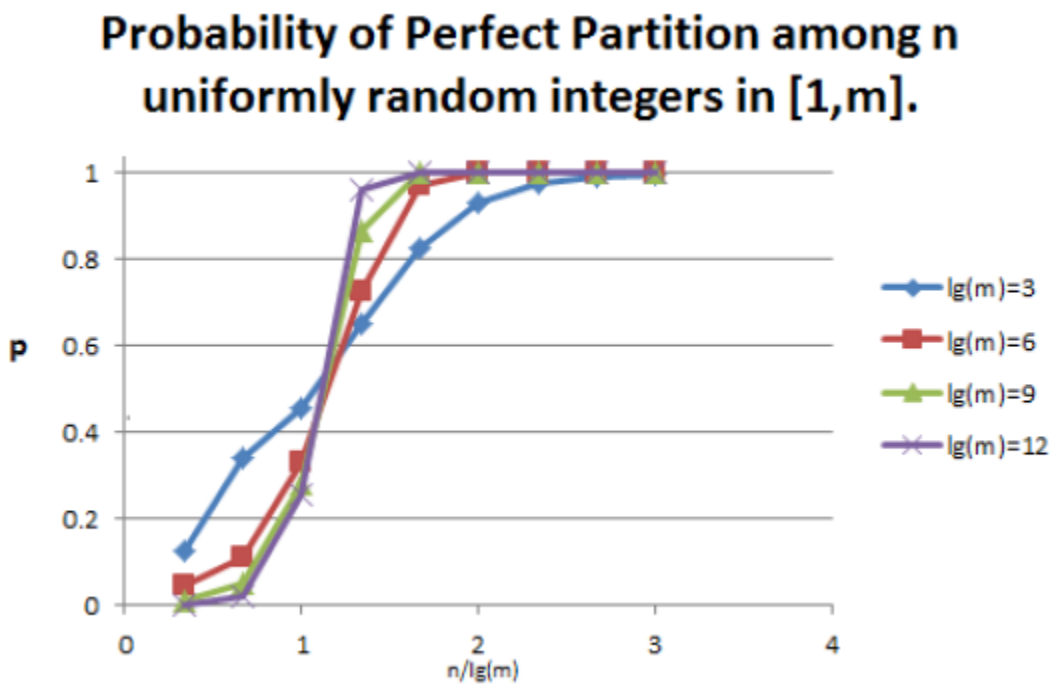


In these cases, the complexity of the search is pruned by either the depth of the search tree in the case of overconstrained instances or the branching factor of each node in the case of underconstrained instances. In contrast, an example of a search tree in an instance near the phase transition would be as follows.

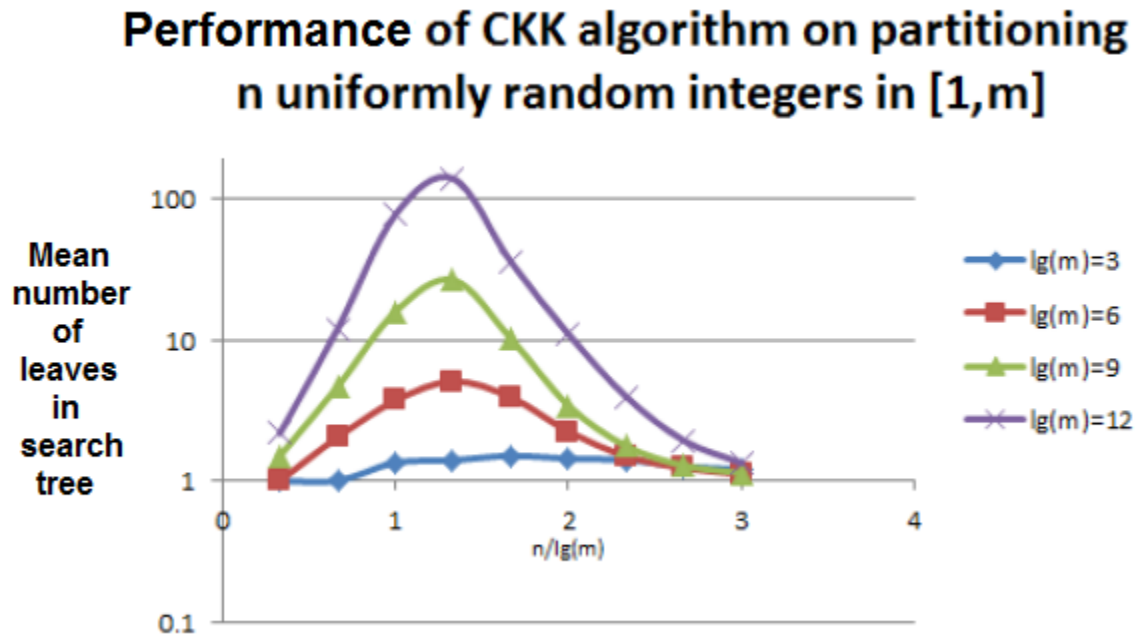


In this case, the search tree is deep due to the abundance of near-solutions and wide due to the relative rarity of both solutions allowing one to end the search or constraints allowing one to prune branches.

In the problem Partition, we are to partition a set of n integers distributed from $[1, m]$. The constrainedness parameter of interest is observed to be $\frac{n}{log(m)}$ [GW96]. In the diagram below, we show, empirically, the probability of a perfect partition (difference of at most one between two subsets) existing among n uniformly random integers among $[1, m]$ with respect to $\frac{n}{log(m)}$, the constrainedness parameter. The data is approximated by running 10000 test cases.



Note the increasing steepness of the transition from almost never having a perfect partition to almost always having a perfect partition as the problem size is scaled, approaching that of a step function. Empirical testing has also demonstrated that most known depth-first search algorithms for NP-complete problems are observed to be increasingly complex in the area of the phase transition. In the following diagram, we are plotting the complexity of invoking the Complete Karmarkar-Karp (CKK) algorithm on the same 10000 instances of Partition of n uniformly random integers among $[1, m]$.



Some NP-complete problems of interest in election systems have also been shown to exhibit phase transitions. In [Wal09], it is shown that manipulation of scoring protocols of three candidates exhibits a phase transition when the constrainedness parameter of $\frac{\sqrt{n}}{m}$ reaches a constant critical value, irrespective of the problem size. The probability of a manipulation existing makes an abrupt change from 0 to 1 for large m as one crosses this phase transition. In other words, $\Theta(\sqrt{n})$ manipulators are asymptotically needed for one to affect the outcome of an election with n established voters. However, unlike the case of other NP-complete problems, Walsh further shows empirically that instances in which finding a manipulation, or showing that none exist, using known algorithms, do not directly correspond to the phase transition of this problem. It is found that, invoking the Complete Karmarkar-Karp (CKK) algorithm on these manipulation instances, finding a manipulation or showing that none exist can be done on average with a constant number of branches of the CKK algorithm. This is true even when the parameters of the election are chosen within the phase transition of the manipulation problem. In other words, empirical testing did not demonstrate an increasing complexity peak, as seen in the above diagram, as one crosses the phase transition with respect to the constrainedness parameter of interest in the problem of manipulation.

Walsh interprets this as that such elections are often easy to manipulate, and that an

exponentially rare subset of hard instances contribute to the worst-case hardness of the manipulation problem. The same phase transition parameter of manipulability, in which the probability of manipulation leaps from almost 0 to almost 1 when the $\frac{\sqrt{n}}{m}$ reaches a critical value, is also demonstrated empirically by Connett [Con10] for other election systems, including those involving multiple rounds. Although the exact constant factor appears different in each election system, in all cases, it requires $\Theta(\sqrt{n})$ manipulators to affect the outcome of an election with n established voters.

Chapter 3

Worst-Case Complexity of Manipulating k-Approval Elections

In this chapter, we evaluate the worst-case computational complexity of manipulation, bribery, and control, in scoring protocols and families of scoring protocols of the form k-approval, k-veto, and some $f(m)$-approval elections. These systems are defined in Section 2.1.

3.1 Table of Results

We summarize the worst-case complexity results of this chapter in the tables below, with new results in bold.[1]

Unweighted Cases

	1-app	2-app	3-app	k-app, $k \geq 4$
Constructive Manipulation	P	P	P	P
Constructive Bribery	P	**P**	**NPC**	**NPC**
Constructive Control by Adding Voters	P	**P**	**P**	**NPC**
Constructive Control by Deleting Voters	P	**P**	**NPC**	**NPC**
Constructive Control by Adding Candidates	NPC	**NPC**	**NPC**	**NPC**
Constructive Control by Deleting Candidates	NPC	**NPC**	**NPC**	**NPC**

[1] In the table, swb is the complexity of Simple Weighted b-Edge Cover of Multigraphs and sbw that of Simple b-Edge Weighted Cover of Multigraphs.

	1-veto	2-veto	3-veto	k-veto, $k \geq 4$
Constructive Manipulation	P	P	P	P
Constructive Bribery	P	**P**	**P**	**NPC**
Constructive Control by Adding Voters	P	**P**	**NPC**	**NPC**
Constructive Control by Deleting Voters	P	**P**	**P**	**NPC**
Constructive Control by Adding Candidates	NPC	**NPC**	**NPC**	**NPC**
Constructive Control by Deleting Candidates	NPC	**NPC**	**NPC**	**NPC**

Weighted Voter Cases

	1-app	2-app	3-app	k-app, $k \geq 4$
Constructive Manipulation	P	NPC	NPC	NPC
Constructive Bribery	P	NPC	NPC	NPC
Constructive Control by Adding Voters	P	?	?	**NPC**
Constructive Control by Deleting Voters	P	**sbw**	**NPC**	**NPC**
Constructive Control by Adding Candidates	NPC	**NPC**	**NPC**	**NPC**
Constructive Control by Deleting Candidates	NPC	**NPC**	**NPC**	**NPC**

	1-veto	2-veto	3-veto	k-veto, $k \geq 4$
Constructive Manipulation	NPC	NPC	NPC	NPC
Constructive Bribery	NPC	NPC	NPC	NPC
Constructive Control by Adding Voters	P	**sbw**	**NPC**	**NPC**
Constructive Control by Deleting Voters	P	?	?	**NPC**
Constructive Control by Adding Candidates	NPC	**NPC**	**NPC**	**NPC**
Constructive Control by Deleting Candidates	NPC	**NPC**	**NPC**	**NPC**

Unweighted $Bribery Cases

	1-app	2-app	3-app	k-app, $k \geq 4$
Constructive $Bribery	P	?	**NPC**	**NPC**

	1-veto	2-veto	3-veto	k-veto, $k \geq 4$
Constructive $Bribery	P	$\geq_p$ **swb**	?	**NPC**

3.2 Misuses in Elections with a Fixed Number of Candidates

In [FHH09], it is shown that bribery of unweighted scoring protocol elections can be solved in polynomial time with brute force when the number of candidates is fixed, since the number of possible preference orderings is also fixed, and that voters with equal preferences are not distinguishable (see the definition of succinct preference representation in Section 2.7). The same principle also applies to control by adding or deleting voters, as well as manipulation. Bribery of fixed scoring protocols is also polynomial-time computable even when voters have prices, since it suffices to bribe only the cheapest voters among those with equal preferences. We review the result for $bribery from [FHH09] below.

Theorem 3.2.1 *[FHH09] α-$bribery is polynomial-time computable for all scoring protocols $\alpha = (\alpha_1, \ldots, \alpha_m)$ with a fixed number of candidates m.*

Proof: Because m is a constant, there are only a constant, $m!$, number of distinct preference profiles. The algorithm given in [FHH09] partitions the set of voters into $m!$ subsets, $V = V_1 \cup \cdots \cup V_{m!}$, with voters in each subset having equal preferences. A bribery can then be described by two sequences of integers, $b_1, \ldots, b_{m!}$, and $d_1, \ldots, d_{m!}$, where each b_i indicates how many voters from V_i we are bribing and d_i indicates how many voters will be given the preferences of V_i. Without loss of generality, we can assume we are bribing the cheapest b_i voters of the set V_i.

Since each bribery can be described with $2m!$ variables, and each variable is bounded $0 \leq b_i, d_i \leq ||V||$, this algorithm tries at most $(||V||+1)^{2m!}$ possible briberies. This quantity is polynomial with respect to the size of the problem instance. Thus, α-$bribery is polynomial-time computable. ■

Also owing to the constant number of possible preference orderings, unweighted scoring protocols of a fixed number of candidates are also computationally vulnerable to manipulation as well as control by adding or deleting voters.

Theorem 3.2.2 *All scoring protocols $\alpha = (\alpha_1, \ldots, \alpha_m)$ are computationally vulnerable to constructive control by either adding or deleting voters, as well as to manipulation for a fixed number of candidates m.*

It is important to note that despite the theoretical polynomial-time computability in this case, in practice, this problem is significantly difficult for moderate-sized m, due to the exponential nature of $m!$. It is often the case that the brute-force algorithm, despite being faster in the worst case, is not the best approach to this problem empirically.

In [HH07, FHH09] it is shown that most nontrivial manipulation and bribery problems in weighted scoring protocol elections are NP-hard even for a fixed number of candidates. We will discuss these cases in Section 3.8.

It should be noted that control by adding or deleting candidates in elections of a constant number of candidates (participating and additional candidates) is easy, as there are a constant number of different sets of candidates to add or delete, and whether or not a distinguished candidate will win or lose the election can be checked in polynomial time.

3.3 Destructive Misuses

To demonstrate the vast differences between the problems of constructive misuse and that of destructive misuse, we prove that unweighted approval-based elections are computationally vulnerable to most forms of destructive misuse. This is true even for families of scoring protocols, in which the size of the candidate set is unbounded. Consider the problem of bribery below.

Theorem 3.3.1 *Unweighted $f(m)$-approval elections are computationally vulnerable to destructive bribery.*

Proof: As a reminder, we assume that $f(m)$ is computable in time polynomial with respect to m.

In a destructive bribery, we want some candidate $p' \neq p$ to beat p. Thus, we want to have p' gain as many points as possible relative to p.

For each $p' \neq p$, we separate the voters into 3 categories:

1. Those approving of p but not of p'.
2. Those approving either both of p and p' or neither of p nor p'.
3. Those approving of p' but not of p.

Clearly, briberies of votes of the first type take priority over those of the second type, as p' can gain two points relative to p. Voters of the third type should not be bribed at all. We check whether p' can potentially beat p by first bribing voters of type one, and then those of type two, up to a total of our quota q. We always bribe voters to approve of p' but not p.

We find such a bribery for each other candidate $p' \neq p$ by bribing q voters using the procedure above. If there exists no candidate $p' \neq p$ that can be made to beat p, we reject. ■

Intuitively, the reason such briberies are easy is that we only need to ensure that p' beats p. It does not matter if other candidates will beat both p' and p as a result of the bribery, as p will lose. This principle is also demonstrated in [Rus07] for the case of Borda elections and [FHH09] for the case of scoring protocols in general.

Theorem 3.3.2 *Unweighted $f(m)$-approval elections are computationally vulnerable to destructive manipulation and destructive control adding or deleting voters.*

Proof: The concept of attempting to make each candidate $p' \neq p$ beat p can be applied to manipulation, as well as control by adding and deleting voters. In manipulation, each manipulator approves of p' but not of p. Similarly, in control by adding voters, we only add voters that approve of p' but not of p. In control by deleting voters, we only delete voters that approve of p but not of p'. ■

We conclude that all unweighted approval-based scoring protocols, as well as families of scoring protocols, $f(m)$-approval, are computationally vulnerable to destructive bribery, manipulation, and control by adding or deleting voters. The case of destructive control by adding or deleting candidates will be considered in Section 3.7. The remainder of this section will thus focus on constructive problems.

3.4 Manipulation of Approval-Based Scoring Protocols

It is shown in [ZPR09] that manipulation in the $f(m)$-approval families of scoring protocols (recall that $f(m)$ is computable in polynomial time with respect to the quantity of m in this case) is polynomial-time computable, by a greedy algorithm. The greedy algorithm operates by iteratively assigning the manipulators' preferences as follows: each manipulator approves p, the distinguished candidate and the current $f(m)-1$ candidates with the lowest scores. It can be shown that this algorithm is correct by induction on the number of manipulators. Such elections are thus computationally vulnerable to insincere voting from a coalition of agents for their collective benefit, whether their goal is to elect a preferred candidate (constructive manipulation) or to prevent the election of a despised candidate (destructive manipulation).

3.5 Bribery in Approval-Based Scoring Protocols

In unweighted and unpriced families of scoring protocols, the goal of the bribery problem is to determine the minimum number of voters one needs to change the preferences for (i.e., bribe) to achieve a desirable outcome. As we have seen earlier, if our goal is to exclude a candidate from winning, as in destructive bribery, this problem is easy and can be computed in polynomial time for all approval-based elections. We thus now focus on constructive bribery cases, where we aim to elect a distinguished candidate.

Theorem 3.5.1 ([FHH09]) *Unweighted* 1*-approval elections are computationally vulnerable to bribery by unpriced voters.*

Proof: In this case, one simply bribes the voters that vote for the current winner to vote for our distinguished candidate, p, until either p wins, in which case we accept, or until we run out of voters to bribe, in which case we reject. ■

Theorem 3.5.2 ([FHH09]) *Unweighted* 1*-veto elections are computationally vulnerable to bribery by unpriced voters.*

Proof: In this case, we simply bribe voters whom veto p to veto for the current winner instead. ■

We now demonstrate how the seemingly unrelated problem of Set Cover relates to bribery of election systems, and prove the resistance to bribery of some approval-based election systems.

In [HHR07] a reduction is given from X3C to control in approval elections by adding or deleting voters. The reduction operates by encoding the elements as candidates and voters as subsets of elements, by approving these elements. A number of buffer candidates and voters are added to enforce the constraints of the problem of X3C.

We demonstrate how to modify this result to the cases of k-approval and k-veto, where either the number of candidates approved or vetoed is fixed, for appropriately valued k. The reduction requires the usage of additional buffer candidates and voters, as in the reduction given in [HHR07], each voter approves a different number of candidates.

Theorem 3.5.3 *Unweighted k-approval elections are computationally resistant to bribery by unpriced voters for $k \geq 3$.*

Proof: We first show resistance for the case of 3-approval elections. Cases for $k > 3$ can be shown by a simple change. We make our reduction from Exact 3-Set Cover (X3C).

Let

$$S = \{s_1, \ldots, s_{3m}\}$$

and

$$T_1 = \{t_{1,1}, t_{1,2}, t_{1,3}\}, \ldots, T_n = \{t_{n,1}, t_{n,2}, t_{n,3}\}$$

be an instance of X3C. Without loss of generality, we assume $n \geq m$.

We construct our k-approval election as follows. Our candidate set will be

$$C = \{p, p', p''\} \cup \{s_1, \ldots, s_{3m}\} \cup \{b_1, \ldots, b_{\frac{3(nm+m-n)}{2}}\}.$$

For each 3-set $T_i = \{t_{i,1}, t_{i,2}, t_{i,3}\}$ we construct a voter who approves of $\{t_{i,1}, t_{i,2}, t_{i,3}\}$. For $1 \leq i \leq \frac{3(nm+m-n)}{2}$ we add voters that approve of b_i and two members of $s_1, \ldots, s_{3m}$. We choose these votes in a way such that each s candidate receives exactly $n+1$ approvals in total. Finally, we add $n-m$ votes that approve of $\{p, p', p''\}$. We set our bribery quota q to m.

In this election, we note that each of p, p', p'' receives $n-m$ approvals. Each of s_1, ..., s_{3m} receives $n+1$ approvals, and each candidate b_i for $1 \leq i \leq \frac{3(nm+m-n)}{2}$ receives one approval.

If there is an exact covering by 3-sets, we note that bribing the agents corresponding to the 3-sets to approve of $\{p, p', p''\}$ instead will allow p to win. On the other hand, a valid bribery must remove one approval of each of s_1, ..., s_{3m}, since the most number of approvals that can be given to p is m. Note that bribing any of the votes corresponding to T_i removes three approvals, collectively, from $s_1, \ldots, s_{3m}$. In contrast, all other bribes will remove at most two approvals, collectively, from $s_1, \ldots, s_{3m}$. Thus, a successful bribery of $\leq m$ votes can only involve votes corresponding to T_i, and each candidate of $s_1, \ldots, s_{3m}$ must lose one approval. This corresponds to an exact covering.

A simple extension to show that k-approval elections for $k > 3$ are also computationally resistant to bribery can be constructed by adding additional buffer candidates, such that each voter approves of distinct buffer candidates in addition to the three edges elements in the subset. Each buffer candidate would thus receive only one approval, and cannot influence the election. ■

A similar reduction, in the next theorem shows this hardness result for k-veto elections.

Theorem 3.5.4 *Unweighted k-veto elections are computationally resistant to bribery by un-*

priced voters for $k \geq 4$.

Proof: Again, we start with 4-veto elections and demonstrate how our reduction can be extended.

Let

$$S = \{s_1, \ldots, s_{3m}\}$$

and

$$T_1 = \{t_{1,1}, t_{1,2}, t_{1,3}\}, \ldots, T_n = \{t_{n,1}, t_{n,2}, t_{n,3}\}$$

be an instance of X3C. In this reduction, we will restrict the X3C instance to cases in which $n \geq 2m$, for reasons which will be clear later.

We construct a 4-veto election of $3m + 5$ candidates,

$$C = \{p\} \cup \{b_1, b_2, b_3, b_4\} \cup \{s_1, \ldots, s_{3m}\}.$$

For each set $T_i = \{t_{i,1}, t_{i,2}, t_{i,3}\}$, we have one voter who veto $\{p, t_{i,1}, t_{i,2}, t_{i,3}\}$. We add $\frac{mn-m^2+m-3n}{4}$ voters whom veto four candidates of $\{s_1, \ldots, s_{3m}\}$ such that each s_i receives exactly $n-m+1$ vetoes. Initially, $n-2m$ voters veto the four buffer candidates, $\{b_1, b_2, b_3, b_4\}$. Under this construction, p initially has n vetoes, each s_i has $n - m + 1$ vetoes, and each buffer candidate has $n - 2m$ vetoes. We set our quota for bribing voters to m.

Consider an exact covering of S. Bribing the votes corresponding to the sets to vote for the 4 buffer candidates instead will leave p with $n-m$ vetoes, each s_i with $n-m$ vetos, and each buffer candidate with $n - m$ vetoes. p is clearly made a winner. Conversely, consider any bribery of at most m voters such that p is made a winner. In any bribery of at most m voters, p must retain at least $n - m$ vetoes. If p is made a winner, each s_i must retain at least $n - m$ vetoes, and can lose at most one veto. Since there are only $3m$ such candidates and only votes corresponding to the 3-sets veto 3 such candidates, these are the only votes we can bribe in such a bribery. Thus, this corresponds to an exact covering.

This reduction can also be extended to k-veto elections for all $k > 4$, by adding k buffer candidates in total. ■

In the next result, we see how a slightly modified form of the greedy algorithm introduced in [FHH09] can be used to bribe 2-veto elections.

Theorem 3.5.5 *Unweighted 2-veto elections are computationally vulnerable to bribery by unpriced voters.*

Proof: Consider an election $E = (C, V)$ and quota $q > 0$.

Without loss of generality, we will only bribe voters that veto p. If no voters veto p, then p is a winner and we can accept. Similarly, if p has at most q vetoes, then we can bribe these voters and make p a winner. Thus, we will assume that p has more than q vetoes. Also without loss of generality, we will not bribe any voters to veto p.

For each candidate $c \in C$, let Vetoes(c) be the number of vetos currently received by candidate c, and for $c_1 \neq c_2$, Vetoes(c_1, c_2) be the number of voters who veto both c_1 and c_2. Then, following our bribery of q voters, p will receive exactly Vetoes(p) $- q$ vetoes. Since we don't want any candidate to have fewer than this many vetoes, for each $c \neq p$, we can bribe up to max(Vetoes(p, c), Vetoes(p) $-$ Vetoes(c) $- q$) voters whom veto both p and c and ensure that c will not beat p.

We will thus bribe a total of

$$\max(\sum_{c \neq p} \max(\text{Vetoes}(p, c), \text{Vetoes}(p) - \text{Vetoes}(c) - q), q)$$

voters. The votes chosen does not matter, since we are removing vetoes from candidates who will not beat p, and we are bribing the maximum number of votes feasible. We must then assign these votes such that each candidate currently beating p receives enough vetoes to lose to p. This is similar to the polynomial-time computable problem of 2-veto manipulation. ■

The next two results, of bribery of 2-approval and 3-veto elections, demonstrate the connection between the seemingly unrelated problem of Simple b-Edge Cover and manipulating election systems. We will see in this and other results that this connection occurs most often when the election system distinguished two candidates from the remaining candidates. This is because the connection equates candidates to vertices of graphs and voters to edges, with the two candidates being distinguished by each voter equated with the two vertices connected by the corresponding edge.

Theorem 3.5.6 *Unweighted 2-approval elections are computationally vulnerable to bribery by unpriced voters.*

Proof: We solve 2-approval bribery using Simple b-Edge Cover.

Consider an election $E = (C, V)$. We wish to ensure the victory of p by bribing at most q voters.

We first observe that, without loss of generality, we will only bribe voters that do not approve of p. If there are no more than q such voters, then all voters will approve of p and p

will win. Thus, without loss of generality, we assume that more than q voters do not approve of p. Also without loss of generality, we can also assume that we give p one approval for each of these bribed votes, and that we will bribe exactly q voters.

For each candidate $c \in C$, let $\text{app}(c)$ be the number of voters currently approving candidate c, and for $c_1 \neq c_2$, $\text{app}(c_1, c_2)$ be the number of voters who approve both c_1 and c_2. Following bribery, p will receive $\text{app}(p) + q$ approvals. Further, for each candidate $c \neq p$, define $\text{def}(c) = \text{app}(p) + q - \text{app}(c)$. If $\text{def}(c) > 0$, we can give $\text{def}(c)$ approvals to candidate c. If $\text{def}(c) < 0$, we must bribe $-\text{def}(c)$ voters who currently approve of c.

Let $C_1 = \{v \mid \text{def}(c) \geq 0\}$ and $C_2 = \{v \mid \text{def}(c) < 0\}$. Thus, for each $c \in C_2$, we must bribe at least $-\text{def}(c)$ voters who currently approve of c. Suppose that we will bribe exactly s_1 voters approving one voter in C_2 and s_2 voters approving two voters in C_2. We will explain what s_1 and s_2 will be later.

Let $E = \sum_{c \neq p} \max(0, -\text{def}(c))$. This is the total number of "excess" approvals that must be bribed from each nondistinguished candidates. If $E > 2q$, then clearly bribery is not possible, since a bribery of q voters can only remove $2q$ excess approvals. Conversely, let $D = \sum_{c \neq p} \max(0, \text{def}(c)) + s_1$. This is the total "deficit" approvals that we can give to the nondistinguished candidates; initially, we can give $\sum_{c \neq p} \max(0, \text{def}(c))$ approvals to the candidates of C_1. We can give s_1 more approvals due to the bribery of voters approving candidates in C_1.

Thus, we can bribe no more than D voters, since each voter will have to approve one nondistinguished candidate in addition to p (Note that without loss of generality, we will not bribe voters who have a positive deficit, and thus, at most D approvals may be given to those candidates). Thus if $q < D$, then bribery is not possible with this set of parameters.

Construct the multigraph G as follows.

Let $V(G) = C_2 \cup \{x\}$ and define $E(G)$ as follows. For every voter approving $\{u, v\} \subseteq C_2$, we add an edge (u, v) and for each voter approving $u \in C_2$ and $v \in C_1$, we add an edge (u, x). We set the b-values to $b_c = -\text{def}(c)$ for $c \in C_2$ and $b_x = s_1$. Note that in essence, x represents the candidates of C_1 in this construction.

Claim: Given that $D \leq q \leq \frac{E}{2}$, G has a simple b-edge covering of q edges iff there exists a bribery of q voters, of which s_1 of the bribed voters approve one candidate of C_2 and s_2 approve two candidates of C_2, making p a winner.

Consider a simple b-edge covering of q edges. Without loss of generality, we may assume x is covered by exactly $b_x = s_1$ edges, as we may modify the covering otherwise. We bribe the voters corresponding to the q edges to approve of p and one nondistinguished candidate.

In the case of edges linking $v \in G$ and x, it suffices to bribe any voter approving v and a candidate in C_1. Since $q \leq D$, it is possible to give the latter approval to the candidates in C_1 without exceeding the score of p. As $b(v)$ corresponds to the number of approvals that must be removed from each $v \in C_2$, p will beat each candidate in C_2.

Conversely, consider a bribery of voters electing p satisfying the paramters. For each candidate $c \in C_2$, at least $-\text{def}(c)$ voters approving c must be bribed. This corresponds to a b-edge cover. Also, by the definition of the parameters, exactly s_1 voters approving candidates in C_1 will be bribed. This corresponds to a solution of the b-Edge Cover problem above.

To find a bribery electing p, we enumerate the values of s_1 and s_2 such that $s_1 + s_2 = q$ and attempt the corresponding construction of b-Edge Cover. ■

Theorem 3.5.7 *Unweighted* 3*-veto elections are computationally vulnerable to bribery by unpriced voters.*

Proof: We observe that, instead of bribing voters not approving p, we bribe voters vetoing p. A similar reduction from this problem to that of b-Edge Cover can be constructed by converting votes vetoing candidates p, c_1, and c_2 into an edge between c_1 and c_2 in our graph. ■

3.6 Controlling an Election via Voters

The chair of an election can control the outcome by controlling the candidate set. Three methods of candidate control of interest are, adding voters to the voter set, removing voters from the voter set, and partitioning the voters and conducting subelections. An example of the latter case can be seen in the Electoral Colleges in the Presidential Elections of the United States, where a separate plurality election is conducted within each state. Throughout the history of the United States, minorities, women, and persons under the age of 21 have also been added to the voter set of the Presidential Election.

We evaluate the computational complexity of such controls in families of approval-based scoring protocols.

Theorem 3.6.1 *Unweighted* 1*-approval (i.e., plurality) and* 1*-veto (i.e., veto) are computationally vulnerable to control by adding or deleting voters.*

Proof: All of these problems can be solved in polynomial time by simple greedy algorithms. To control by adding voters for plurality elections, we simply add votes that approve p until either p wins, in which case we accept, or we are out of votes to add or have added our quota, in which we case reject, as it cannot benefit p to add votes approving another candidate. To control by deleting voters, for each candidate p' beating p, we must delete votes approving p' until p' has as many approvals as p. In veto elections, to control by adding voters, for each candidate p' beating p, we must add votes vetoing p' until it has as many vetoes as p. To control by deleting voters, we delete voters vetoing p. ■

Theorem 3.6.2 *Unweighted* 2*-approval elections are computationally vulnerable to control by adding voters, and unweighted* 2*-veto elections by deleting voters.*

Proof: Consider the case of control by adding voters in 2-approval elections. Without loss of generality, we only need to consider adding voters which approve of p, and consider adding as many voters as possible. This determines the final score of p, allowing one to determine if it is possible to choose votes such that no other candidates will exceed this total.

In the case of 2-veto, we only need to consider deleting voters who veto p. We compute the number of vetoes p will retain, and determine if it is possible to delete these voters such that no other candidate will have fewer vetoes. ■

Theorem 3.6.3 *Unweighted* 3*-approval and* 2*-veto elections are computationally vulnerable to control by adding voters, and unweighted* 2*-approval and* 3*-veto elections by deleting voters.*

Proof: As demonstrated in the case of bribery, we represent the candidates as vertices and the voters by edges. The reduction is to Simple b-Edge Matching of Multigraphs, in this case.

Consider the case of control by adding at most q unweighted voters in 3-approval elections. Without loss of generality, we will add exactly q voters, all of which approve p. The final score of p is thus $s(p) + q$, where $s(p)$ is the initial score of p. For each candidate $c \in p$, we may add at most $s(p) + q - s(c)$ voters approving c.

Each voter that is added also approves two other candidates. In this case, each voter corresponds to an edge between the two other candidates, and we must find q edges such that each vertex corresponding to the candidate c is covered by at most $s(p)+q-s(c)$ edges, giving us q voters that can be added without exceeding this score. This corresponds to the problem of Simple b-Edge Matching of Multigraphs, which is polynomial-time computable.

A similar reduction will also show that 3-veto elections are vulnerable to control by deleting voters. ■

Theorem 3.6.4 *k-approval elections are computationally resistant to constructive control by adding voters for $k \geq 4$.*

Proof: We begin by showing that 4-approval control by adding voters is NP-hard and extend it to k-approval for $k > 4$.

Let

$$S = \{s_1, \ldots, s_{3m}\}$$

and

$$T_1 = \{t_{1,1}, t_{1,2}, t_{1,3}\}, \ldots, T_n = \{t_{n,1}, t_{n,2}, t_{n,3}\}$$

be an instance of X3C.

Without loss of generality, we can assume that 4 divides $3m$, by adding at most 4 dummy sets to our instance. We define our candidate set

$$C = \{p\} \cup \{s_1, \ldots, s_{3m}\}.$$

We assign votes to V such that initially p receives no approvals and s_i receives $m-1$ approvals for each $1 \leq i \leq 3m$. Note that this construction is possible provided that 4 divides $3m$. There are thus $3m + 1$ candidates and $\frac{3m(m-1)}{4}$ voters.

For each set $T_i = \{t_{i,1}, t_{i,2}, t_{i,3}\}$ in our instance of X3C, we add an unestablished voter v_i to V' which approves of $\{p, t_{i,1}, t_{i,2}, t_{i,3}\}$. We show that our instance of X3C has an exact covering if and only if there exists a control of this election by adding m new voters.

Consider an exact 3-set covering. Adding the votes that correspond to the m 3-sets of this covering would add one approval to each of s_i and m approvals to p, giving each candidate exactly m approvals. Thus, p becomes a winner. Conversely, consider a successful control that adds at most m new voters and makes p a winner. Since each s_i candidate currently has $m - 1$ approvals and all votes approve of at least one such candidate, we see that we must add exactly m new voters to potentially make p a winner. However, in such a case, p will end up with m approvals, so none of the s_i candidates can receive two new approvals, as this would give that candidate $m + 1$ approvals. This corresponds to a solution to the X3C instance.

In the case of k-approval control by adding voters for $k > 4$, we modify the reduction as follows. Each voter in V now approves of k candidates in S, and thus without loss of generality, we now assume that k divides $3m$. In each voter v_i in V', we will approve of $k - 4$ buffer candidates, $\{b_1, \ldots, b_{k-4}\}$ in addition to $\{p, t_{i,1}, t_{i,2}, t_{i,3}\}$. Following an addition

of voters, each buffer candidate will have exactly the same number of approvals as p, and will this not affect the status of p. ■

Theorem 3.6.5 *k-veto elections are computationally resistant to constructive control by adding voters for $k \geq 3$.*

Proof: We begin by showing that 3-veto control by adding voters is NP-hard and extend it to k-veto for $k > 3$. Let

$$S = \{s_1, \ldots, s_{3m}\}$$

and

$$T_1 = \{t_{1,1}, t_{1,2}, t_{1,3}\}, \ldots, T_n = \{t_{n,1}, t_{n,2}, t_{n,3}\}$$

be an instance of X3C. We define our candidate set to be

$$C = \{p, p', p''\} \cup \{s_1, \ldots, s_{3m}\}.$$

Our initial voter set V will consist of one vote, which vetoes $\{p, p', p''\}$.

For each set $T_i = \{t_{i,1}, t_{i,2}, t_{i,3}\}$, we add a voter v_i to our unestablished voter set V' who vetoes $\{t_{i,1}, t_{i,2}, t_{i,3}\}$. We show that a manipulation by adding at most m voters exists iff there is an exact covering for this instance of X3C.

Consider an exact 3-set covering. Adding the m votes corresponding to the covering gives each candidate exactly one veto. p is thus made a winner. In contrast, p can only be made a winner by adding one veto to each candidate in s_i. Since only m votes are added, this corresponds to a solution of the X3C instance. ■

Theorem 3.6.6 *k-approval elections are computationally resistant to constructive control by deleting voters for $k \geq 3$.*

Proof: The reduction for this case is similar to that of bribery. In this case, instead of bribing the voters to vote for $\{p, p', p''\}$ we simply delete them. ■

Theorem 3.6.7 *k-veto elections are computationally resistant to constructive control by deleting voters for $k \geq 4$.*

Proof: We note that the construction in the proof that k-veto is computationally resistant to bribery in the unweighted and unpriced case also applies to constructive control by deleting voters. In this case, instead of bribing the voters to vote for buffer candidates, we simply delete them. ■

3.7 Controlling an Election via Candidates

The chair of an election can also attempt to influence the outcome by affecting the voter set, dictating rules for candidates that may or may not participate in the election. A common example is that of cloning, in which a winning candidate is made worse off by the addition of a similar candidate, causing the votes to be split among them, and possibly allowing a third independent candidate to win.

We briefly review the known complexities of this problem given in [BTT92] and [HHR07] and demonstrate how to generalize it to other cases of approval elections. Control by adding and deleting candidates is hard in both the constructive and destructive case in two of the simplest families of scoring protocols: plurality and veto. The reductions given in [BTT92, HHR07] are from Hitting Set.

3.7.1 Destructive Control

In [HHR07], a construction was given demonstrating that destructive control via either adding or deleting candidates in a plurality election is NP-hard, via a reduction from Hitting Set. We demonstrate how to generalize this reduction to the cases of k-approval by adding buffer candidates, and then to k-veto by making some modifications. This hardness result for adding candidates is also shown independently in [EFS10].

Theorem 3.7.2 *k-approval elections are computationally resistant to destructive control by either adding or deleting candidates.*

Proof: Consider an instance of Hitting Set, where we are given a set

$$S = \{s_1, \ldots, s_m\},$$

n subsets of S, $T_1, \ldots, T_n$, and positive integer $1 \leq q \leq m$.

In our election, the candidate set will consists of

$$\begin{aligned} C = \{c, c'\} \quad &\cup \quad S \\ &\cup \quad \{x^1_{i,j} \mid 1 \leq i \leq 2(m-q) + 2n(q+1) + 4, 1 \leq j \leq k\} \\ &\cup \quad \{x^2_{i,j} \mid 1 \leq i \leq 2n(q+1) + 5, 1 \leq j \leq k\} \\ &\cup \quad \{x^3_{i,j,\ell} \mid 1 \leq i \leq n, 1 \leq j \leq 2(q+1), 1 \leq \ell \leq k\} \\ &\cup \quad \{x^4_{i,j,\ell} \mid 1 \leq i \leq m, 1 \leq j \leq 2, 1 \leq \ell \leq k\}. \end{aligned}$$

For each $1 \leq i \leq 2(m-q) + 2n(q+1) + 4$ we add a voter with preferences

$$c \succ x^1_{i,1} \succ \cdots \succ x^1_{i,k-1} \succ c' \succ \cdots,$$

and for each $1 \leq i \leq 2n(q+1) + 5$ we add a voter with preferences

$$c' \succ x^2_{i,1} \succ \cdots \succ x^2_{i,k-1} \succ c \succ \cdots.$$

Thus, there are, just as in [HHR07], $2(m-q) + 2n(q+1) + 4$ voters preferring c to all other candidates, and $2n(q+1) + 5$ candidates preferring c'.

For each $1 \leq i \leq n$ and $1 \leq j \leq 2(q+1)$, we add one voter with preferences

$$T_i \succ x^3_{i,j,1} \succ \cdots \succ x^3_{i,j,k-1} \succ c \succ \cdots.$$

For each $1 \leq i \leq m$ and $1 \leq j \leq 2$, we add one voter with preferences

$$b_i \succ x^4_{i,j,1} \succ \cdots \succ x^4_{i,j,k-1} \succ c' \succ \cdots.$$

The above construction is similar to that given in [HHR07]. We note that in this case, we must ensure that we do not delete the buffer candidates in our case for deleting candidates. We thus add the following voters.

For each $1 \leq i \leq 2(m-q) + 2n(q+1) + 4$, one voter with preferences

$$x^1_{i,1} \succ \cdots \succ x^1_{i,k} \succ c \succ \cdots$$

and for $1 \leq i \leq m$ and $1 \leq j \leq 2$, one voter with preferences

$$x^4_{i,j,1} \succ \cdots \succ x^4_{i,j,k} \succ c \succ \cdots .$$

This ensures that c' cannot gain in relation to c by deleting any buffer candidates.

We show that there is an addition of at most q voters to $(\{c, c'\} \cup B, V)$ such that c is excluded from winning if and only if there exists a hitting set of S of at most q elements. Let S' be a hitting set of S of size q. In the election $(S' \cup B \cup \{c, c'\}, V)$, c receives $2(m - k) + 2n(q + 1) + 4$ approvals, c' receives $2(m - k) + 2n(q + 1) + 5$ approvals, each candidate $s_i \in S$ receives at most $2n(q + 1) + 2$ approvals, and each buffer candidate receives at most 2 approvals. Thus, c' wins this election and c is excluded from winning.

In contrast, let D be a subset, of size at most q, of S such that c is not a winner of $(D \cup B \cup \{c, c'\}, V)$. We first note that if $b \in S$ or b is a buffer candidate, that c beats b. Thus, if c is excluded from winning this election, then c' must beat c. In $(D \cup \{c, c'\}, V)$, c' receives $2n(q+1)+5+2(m-||D||)$ approvals and c receives $2(m-q)+2n(q+1)+4+2(q+1)m'$ approvals, where m' is the number of sets in S that are not hit by D. Since c is excluded from winning, $2(m - q) + 2(q + 1)m' \leq 2(m - ||D||)$, which implies $(q + 1)m' + ||D|| - q \leq 0$. So $m' = 0$ and $||D||$ corresponds to a hitting set of S of size at most q.

As in [HHR07], this reduction also shows that k-approval is computationally resistant to destructive control by deleting voters. In this case, we start with the election $(\{c, c'\} \cup B \cup S, V)$. Destructive control by deleting at most $m - q$ candidates is possible if and only if there exists a hitting set of S of at most q elements. We note here that the additional voters approving the buffer candidates to c is required in this case to prevent one from deleting buffer candidates. In this case, deleting buffer candidates would be useless because c would gain one approval while c' would gain at most one approval. ■

Theorem 3.7.3 *k-veto elections are computationally resistant to destructive control by either adding or deleting candidates.*

Proof: Consider an instance of Hitting Set, where we are given a set $S = \{s_1, \ldots, s_m\}$, n subsets of S, $T_1, \ldots, T_n$, and positive integer $1 \leq q \leq m$.

In our election, the candidate set will consists of

$$
\begin{aligned}
C = \{c, c'\} \quad &\cup \quad S \\
&\cup \quad \{x^1_{i,j} \mid 1 \leq i \leq 2(m-q) + 2n(q+1) + 4, 1 \leq j \leq k\} \\
&\cup \quad \{x^2_{i,j} \mid 1 \leq i \leq 2n(q+1) + 5, 1 \leq j \leq k\} \\
&\cup \quad \{x^3_{i,j,\ell} \mid 1 \leq i \leq n, 1 \leq j \leq 2(q+1), 1 \leq \ell \leq k\} \\
&\cup \quad \{x^4_{i,j,\ell} \mid 1 \leq i \leq m, 1 \leq j \leq 2, 1 \leq \ell \leq k\}.
\end{aligned}
$$

For each $1 \leq i \leq 2(m-q) + 2n(q+1) + 4$ we add a voter with preferences

$$\cdots \succ c \succ x^1_{i,1} \succ \cdots \succ x^1_{i,k-1} \succ c'$$

and for each $1 \leq i \leq 2n(q+1) + 5$ we add a voter with preferences

$$\cdots \succ c' \succ x^2_{i,1} \succ \cdots \succ x^2_{i,k-1} \succ c.$$

For each $1 \leq i \leq n$ and $1 \leq j \leq 2(q+1)$, we add one voter with preferences

$$\cdots \succ c' \succ x^3_{i,j,1} \succ \cdots \succ x^3_{i,j,k-1} \succ T_i.$$

For each $1 \leq i \leq m$ and $1 \leq j \leq 2$, we add one voter with preferences

$$\cdots \succ c \succ x^4_{i,j,1} \succ \cdots \succ x^4_{i,j,k-1} \succ b_i.$$

We have thus, in essence, interchanged the role of approving c with disapproving c' and vice versa, as they have the same effect in relation to the candidates. We must now, as in the above case, add some voters to prevent our reduction from cheating by deleting buffer candidates.

For each $1 \leq i \leq 2(m-q) + 2n(q+1) + 4$, we add one voter with preferences

$$\cdots \succ c' \succ x^1_{i,1} \succ \cdots x^1_{i,k}$$

and for $1 \leq j \leq 2$, one voter with preferences

$$\cdots \succ c' \succ x^4_{i,j,1} \succ \cdots \succ x^4_{i,j,k}.$$

Using the same argument as above, we see that hitting sets in S correspond to valid destructive controls by either adding or deleting candidates. ■

3.7.4 Constructive Control

We examine the reduction given in [BTT92] for constructive control by adding or deleting candidates in plurality elections and generalize it to cases of k-approval and k-veto elections.

In [BTT92], the construction involved an election with established candidates c, c', and d, and unestablished candidates corresponding to the elements of S. The idea was that by adding new candidates helps c gain votes relative to c', but also helps d gain votes relative to c. Thus, candidate d enforces our limit of adding voters, q.

In a constructive control by deleting candidates, there are a number of challenges to overcome in the addition of buffer voters and candidates. First, we must ensure that we cannot cheat by deleting either c' or d. One way to overcome this is to clone both c' and d, making copies of these candidates, all with the same function, such that we cannot delete all of the copies of either. Also, to extend to k-approval or k-veto by adding buffer candidates, we must also ensure that a valid control cannot involve deleting such buffer candidates. This can also be ensured by cloning each of the buffer candidates.

Theorem 3.7.5 *k-approval elections are computationally resistant to constructive control by adding or deleting candidates.*

Proof: Consider an instance of Hitting Set, where we are given a set $S = \{s_1, \ldots, s_m\}$, n subsets of S, $T_1, \ldots, T_n$, and positive integer $1 \leq q \leq m$.

In our election, the candidate set will consist of

$$\begin{aligned} C = \{c\} \quad &\cup \quad \{c'_1, \ldots, c'_{m-q+1}\} \\ &\cup \quad \{d_1, \ldots, d_{m-q+1}\} \\ &\cup \quad S \\ &\cup \quad \{x_i \mid 1 \leq i \leq k-1\} \end{aligned}$$

and the voter set will consist of

$$
\begin{aligned}
V \;=\; & \{(c \succ c'_1 \succ \cdots \succ c'_{k-1} \succ \cdots), (c'_k \succ \cdots \succ c'_{2k-1} \succ \cdots), \ldots, \\
& (c'_{m-q-k+2} \succ \cdots \succ c'_{m-q+1} \succ \cdots) \mid 1 \le i \le (n+2m)(m-q+1) - m + q\} \\
\cup\; & \{(d_1 \succ \cdots \succ d_k \succ \cdots), (d_{k+1} \succ \cdots \succ d_{2k} \succ \cdots), \ldots, \\
& (d_{m-q-k+1} \succ \cdots \succ c'_{m-q+1} \succ \cdots) \mid 1 \le i \le (n+2m)(m-q+1)\} \\
\cup\; & \{(T_i \succ x_1 \succ \cdots \succ x_{k-1} \succ c'_j \succ \cdots) \mid 1 \le i \le n, 1 \le j \le m-q+1\} \\
\cup\; & \{(s_i \succ x_1 \succ \cdots \succ x_{k-1} \succ c \succ \cdots) \mid 1 \le i \le m\} \\
\cup\; & \{(s_i \succ x_1 \succ \cdots \succ x_{k-1} \succ c'_j \succ \cdots) \mid 1 \le i \le m, 1 \le j \le m-q+1\}.
\end{aligned}
$$

As seen in [BTT92], adding a set of candidates corresponding to a hitting set of the established candidates $C - S$ makes c a winner and vice versa, as this is the same election with additional buffer candidates and cloned candidates. In this case, each buffer candidate receives $(n+2m)(m-q+1)$ approvals, and cannot prevent c from winning via any addition of candidates, as c receives at least $(n+2m)(m-q+1)$ approvals.

We show that this is also a valid reduction showing that k-approval is computationally resistant to constructive control by deleting candidates. Consider a deletion of $m-q$ candidates from C. We see that at least one candidate c'_i and d_j must remain. Since the scores of each such candidate will remain the same after deletion, without loss of generality, we can assume that none of the candidates c'_i or d_j are deleted, since they either all win or all lose to c. Also without loss of generality, we can see that deleting any of the buffer candidates will not benefit c in relation to c'_i or d_j more so than deleting an s_j candidate, since it will remove one of the first k candidates from the voters corresponding to each T_i. Thus, if there exists a deletion of $m-q$ candidates of C making c a winner, there also exists a deletion of $m-q$ candidates of S making c a winner. This corresponds to the complement of a hitting set of S. ∎

Theorem 3.7.6 *k-veto elections are computationally resistant to constructive control by adding or deleting candidates.*

Proof: We modify the construction above. The candidate and voter set in this case is defined as follows.

$$\begin{aligned} C = \{c\} \;&\cup\; \{c'_1, \ldots, c'_{m-q+1}\} \\ &\cup\; \{d_1, \ldots, d_{m-q+1}\} \\ &\cup\; S \\ &\cup\; \{x_i \mid 1 \leq i \leq k-1\} \end{aligned}$$

$$\begin{aligned} V \;=\; & \{(\cdots \succ c \succ c'_1 \succ \cdots \succ c'_{k-1}), (\cdots \succ c'_k \succ \cdots \succ c'_{2k-1}), \ldots, \\ & (\cdots \succ c'_{m-q-k+2} \succ \cdots \succ c'_{m-q+1}) \mid 1 \leq i \leq (n+2m)(m-q+1)\} \\ \cup\; & \{(\cdots \succ d_1 \succ \cdots \succ d_k), (\cdots \succ d_{k+1} \succ \cdots \succ d_{2k}), \ldots, \\ & (\cdots \succ d_{m-q-k+1} \succ \cdots \succ c'_{m-q+1}) \mid 1 \leq i \leq (n+2m)(m-q+1) - m + q\} \\ \cup\; & \{(\cdots \succ c \succ x_1 \succ \cdots \succ x_{k-1} \succ T_i) \mid 1 \leq i \leq n\} \\ \cup\; & \{(\cdots \succ c \succ x_1 \succ \cdots \succ x_{k-1} \succ s_i) \mid 1 \leq i \leq m\} \\ \cup\; & \{(\cdots \succ c'_j \succ x_1 \succ \cdots \succ x_{k-1} \succ s_i) \mid 1 \leq i \leq m, 1 \leq j \leq m-q+1\} \end{aligned}$$

■

3.8 Weighted and Priced Cases of Election Misuse

We examine the problem of manipulation, bribery, and control for the case of weighted elections and voters with price tags for bribery. In weighted elections, each voters is given a weight and the points assigned to each voter is scaled by that weight. In some bribery

problems, there is a cost associated with bribing each voter, and we would like to find the cheapest bribery. We would like to determine the effect on the complexity of manipulation.

In [HH07] it is shown that a scoring protocol $\alpha = (\alpha_1, \ldots, \alpha_m)$ is computationally vulnerable to manipulation by weighted voters if and only if $\alpha_2 = \ldots = \alpha_m$ (i.e., basically, only for elections with behavior similar to plurality and the trivial system in which each candidate receives the same score). Thus, $f(m)$-approval weighted elections can be computationally vulnerable to manipulation if and only if $f(m) = 1$ for all $m \geq 1$. We conclude that 1-approval (i.e., plurality) is the only approval-based scoring protocol that is computationally vulnerable to manipulation.

In bribery problems, voters can have both weights and prices. In this case, we pay a price for bribing each voter, and we have a budget q. We may bribe as many votes as we want so long as this budget is not exceeded.

In [FHH09] it is seen that weighted scoring protocols elections are computationally vulnerable to bribery iff $\alpha_2 = \ldots = \alpha_m$, and vulnerable to \$bribery iff $\alpha_1 = \ldots = \alpha_m$. Thus, 1-approval is also the only weighted approval-based scoring protocol that is computationally vulnerable to bribery. No nontrivial weighted approval-based scoring protocol are computationally vulnerable to \$bribery.

Theorem 3.8.1 1-*approval and* 1-*veto (i.e., plurality and veto) elections are computationally vulnerable to constructive \$bribery.*

Proof: The case of \$bribery for 1-approval is shown in [FHH09]. In 1-veto, as in the unweighted case, we must bribe voters vetoing p and give the vetoes to the candidate with the fewest vetoes. Since the voters are indistinguishable by weight, we simply bribe the cheapest voters available. ■

Theorem 3.8.2 *Simple Weighted b-Edge Cover of Multigraphs is polynomial-time many-one reducible to \$bribery in* 2-*veto elections.*

Proof: Consider an instance of Simple Weighted b-Edge Cover of Multigraphs: Let $G = (V, E)$ be a multigraph, and let $b(v)$ denote the multiplicity that $v \in V$ is to be covered, and $w(e)$ denote the weight of edge $e \in E$. We wish to find a b-edge cover of G with edges of total weight at most q.

We construct a 2-veto election as follows. The voter set will be given by $V \cup \{p, p', b, b'\}$. We will be bribing to elect p. Note that the voter set also includes each vertex of G and three buffer candidates, p', b, and b'.

Initially, there are $||E||$ voters of price $q+1$ who veto $\{p, p'\}$ and $||E||$ voters of price $q+1$ who veto $\{b, b'\}$. For each edge $e = \{v_1, v_2\} \in E$ of weight $w(e)$, let there be one voter of price $w(e)$ who vetoes $\{v_1, v_2\}$. Finally, for each vertex $v \in V$, let there be $||E|| + b(v) - \texttt{deg}(v)$ voters of price $q + 1$ who veto $\{v, b\}$. $\texttt{deg}(v)$ is the degree of vertex v in G.

Initially, p has $||E||$ vetoes, while each of p', b, and b' have at least $||E||$ vetoes. Each $v \in V$ has exactly $||E|| + b(v)$ vetoes.

Consider a weighted b-edge cover of G of weight at most q. It is possible to bribe the voters corresponding to this b-edge cover, at cost at most q, to veto $\{b, b'\}$ instead. On the other hand, consider a bribery of cost at most q which makes p a winner. Clearly, only the voters corresponding to the edges in E can be bribed, as all of the other voters have price $q + 1$. It must be the case that for each $v \in V$, at least $b(v)$ voters are bribed. This corresponds to a b-edge cover.

This shows that \$bribery of 2-veto elections is at least as hard as Simple Weight b-Edge Cover of Multigraphs. However, the complexity of this problem remains an open problem. ■

In the next result, we find two problems of control that are equivalent to another variation of b-Edge Cover we defined as Simple b-Edge Weighted Cover of Multigraphs. In this problem, each edge counts differently toward the multiplicity of the vertex being covered. Edges are, however, unweighted, in that a typical instance will seek a covering by some number, q, of edges.

Theorem 3.8.3 *Control by adding weighted voters in 2-veto elections, and by deleting weighted voters in 2-approval elections, is polynomial-time equivalent to Simple b-Edge Weighted Cover of Multigraphs.*

Proof: Consider the case of weighted 2-veto control by adding voters.

Consider an election $E = (C, V)$. For each candidate $c \in C$, define Score(c) to be the current score of candidate c in E. Without loss of generality, we add only voters that do not veto p. To ensure p's victory, for each candidate $c \neq p$ currently beating p, we must add vetoes of weight totaling at least Score(c) − Score(p).

We construct a graph of vertex set $C - \{p\}$. For each voter v of weight $w(v)$ vetoing candidates $\{a, b\}$, we add an edge of multiplicity $w(v)$ connecting vertices a and b. For each candidate $c \in C - \{p\}$, we assign the b-value of its corresponding vertex $b(c) = \max(\text{Score}(c) - \text{Score}(p), 0)$.

Observe that in a valid covering of at most q vertices, adding these q voters ensures that each candidate $c \neq p$ receives vetoes of total weight at least $\max(\text{Score}(c) - \text{Score}(p), 0)$,

and thus p beats c. Conversely, in any addition of voters, without loss of generality, we may disregard any additional voters whom veto p. Since each candidate $c \neq p$ must receive vetoes of total weight at least $\max(\text{Score}(c) - \text{Score}(p), 0)$ to ensure p's victory, edges corresponding to the additional voters must correspond to a b-Edge cover.

Conversely, consider an instance of Simple b-Edge Weighted Cover of Multigraphs. Let $G = (V, E)$ be a multigraph. For each $v \in V$, let $b(v)$ denote the minimum multiplicity of edges that v is to be covered, and for each edge $e \in E$, let $w(e)$ denote the multiplicity of the edge e.

Let $M = \max_v b(v)$ denote the largest b-value of the vertices. Construct an election over the candidates $\{p\} \cup V$ such that initially p receives M vetoes, and each $v \in V$ receives $M - b(v)$ vetoes. For each $e = (v_1, v_2) \in E$, let there be one unestablished voter of weight $w(e)$ vetoing $\{v_1, v_2\}$.

Consider an addition of at most q voters that elects p. As p cannot gain any vetoes from the addition of the voters, clearly, each $v \in V$ must gain at least $b(v)$ vetoes. This corresponds to a b-Edge Cover of at most q edges. Conversely, adding at most q edges corresponding to a b-Edge Cover adds at least $b(v)$ vetoes to each $v \in V$, giving each $v \in V$ at least M vetoes, and electing p.

A similar construction can be used for weighted 2-approval control by deleting voters. ■

Unfortunately, unlike the unweighted cases, this reduction cannot be easily modified for the case of adding weighted voters of 3-veto elections or deleting weighted voters in 3-approval elections. This is because it is not possible to compute the final score of p following the addition or deletion of voters in these cases. Recall that in unweighted cases, it can be assumed that without loss of generality we will alter as many voters as possible.

Theorem 3.8.4 *k-approval elections for $k \geq 3$ and k-veto for $k \geq 4$ are computationally resistant to \$bribery.*

Proof: This problem is a generalization of unpriced bribery for the same election systems, which is NP-hard. ■

Theorem 3.8.5 *Weighted 1-approval and 1-veto elections are computationally vulnerable to constructive control by adding and deleting voters.*

Proof: We consider the four cases separately. In 1-approval constructive control by adding voters, it is only relevant to add voters approving our distinguished candidate p. We thus

add the heaviest voters approving p, up to our quota or until we are out of such voters. Control is possible iff p wins in this case.

In 1-veto constructive control by adding voters, for each candidate p' beating p, we must eventually add votes vetoing p'. Without loss of generality we can add the heaviest vote. Our algorithm thus works as follows. Until p is winning, for each candidate p' currently beating p, we add the vote vetoing p' with the heaviest weight.

In 1-approval constructive control by deleting voters, we see that for each candidate p' beating p, we must eventually delete a vote approving p'. Without loss of generality we can delete the heaviest vote. Our algorithm thus works as follows. Until p is winning, for each candidate p' beating p, we delete the vote approving p' with the heaviest weight.

In 1-veto constructive control by deleting voters, we delete the heaviest votes vetoing p. ■

Theorem 3.8.6 *Weighted k-approval elections for $k \geq 4$ and k-veto elections for $k \geq 3$ are computationally resistant to control by adding voters. Similarly, weighted k-approval elections for $k \geq 3$ and k-veto elections for $k \geq 4$ are computationally resistant to control by deleting voters.*

Proof: These problems are generalizations of the corresponding unweighted problems for the same election systems, which are NP-hard. ■

3.9 Results and Discussion

These results show the variance of complexity of misuse among different problems in election systems of the form k-approval, k-veto, and $f(m)$-approval, and give a result of complexity for infinitely many scoring protocols of an unbounded number of candidates. There are a few interesting cases: These manipulations can either be easy by a simple greedy algorithm, equivalent to a corresponding variation of b-Edge Cover, which is easy for the unweighted and unpriced cases but unknown for the weighted or priced variations, or hard by reduction from Set Cover. Manipulations involving the candidate sets are always difficult as a result of reductions from Hitting Set. These results show a connection between election manipulation and seemingly unrelated graph theory problems.

Our results demonstrate the strengths and weaknesses of approval-based election systems, and we hope they can lead to further generalizations and possible developments of systems that better resist such attacks. There are, however, six open problems: that of control

by adding voters in 2 and 3-approval elections, control by deleting voters in 2 and 3-veto elections, and $bribery of 2-approval and 3-veto elections. These cases cannot be evaluated using the Edge Cover variants, because, for instance, to bribe voters in 3-veto elections, we must choose between those that distinguish three of the candidates other than p. This cannot be accomplished with Simple Weighted b-Edge Cover of Multigraphs. The problem of X3C also does not appear to reduce to this or the other cases. The complexity of Simple Weighted b-Edge Cover of Multigraphs and Simple b-Edge Weighted Cover of Multigraphs is also left as an open problem. We believe it both problems are likely to be polynomial-time computable, due to the results for other variations of b-Edge Cover and the connections between this problem and linear programming.

It is important to realize that NP-completeness only addresses the worst-case complexity of a given problem, and does not take into consideration the distribution of problems that might be given. Some simple distributions were considered in [Wal09, FKN08], and it may be of interest to characterize the complexity of more interesting and realistic distributions, depending upon the application.

This model also makes the assumption that in a k-approval election, each voter may vote for any combination of the k candidates independently. We know that in practice, most elections do not follow this principle, and voters in most realistic elections have highly correlated preferences (e.g., nearly single-peaked preferences (see [BH06])). In [FHHR11], several otherwise hard problems in election manipulation are shown to be easy when restricted to the case of single-peaked preferences. It is shown that constructive or destructive control by adding or deleting candidates is easy for plurality elections, as well as weighted manipulation in some scoring protocols other than plurality, become easy when the voters (including the manipulators) are restricted to having single-peaked preferences. It may thus be of interest to characterize the complexity properties of manipulation in a more realistic distribution of voter preferences.

In the next two chapters, we will be evaluating the complexity of solving of manipulation problems, including some in approval-based scoring protocols and families of scoring protocols, using the known approaches to the problems of Partition and SAT. In doing so, we want to evaluate if the worst-case complexity barrier of NP-completeness is of applicability to the instances we are interested in, and if not, where the hard instances lie and if this constitutes a weakness.

Chapter 4

Solving Election Manipulation Using Integer Partition Problems

The connection between manipulating weighted elections, particularly of a small set of candidates, and partitioning integers into equal subsets, has long been known, and was first explicitly stated by Walsh. In [Wal09], the complexity of manipulating weighted veto elections of three candidates is evaluated using known algorithms for Partition. Walsh observed empirically that even within the phase transition of the problem of manipulation, it is often easy to find a manipulation or show that none exist using the Complete Karmarkar-Karp (CKK) algorithm for Partition. Among other tests, it is further shown that instances that challenge the CKK algorithm empirically involve highly correlated voters. Walsh makes the conclusion that hard instances of manipulation do not directly correspond to the phase transition of the manipulation problem, and are exceedingly rare. This is in direct contrast with most NP-complete problems such as SAT and Partition, which have empirically been shown to be exceedingly difficult to compute for cases near the phase transition with all known backtracking algorithms.

In this chapter, we will be using the best-known algorithms for k-Way Partition, given by Korf [Kor98, Kor09, Kor10], to solve the problem of manipulation in scoring protocols. In particular, we will be introducing naturally analogous algorithms to the problem of k-Way Permutation Partition and demonstrating how the problem of manipulation in elections of scoring protocols can be encoded into instances of this problem. We can then use this reduction to empirically evaluate the complexity of manipulating elections using algorithms for k-Way Permutation Partition. We are especially interested in evaluating the problem of manipulation in scoring protocols, and particularly cases near the phase transition, to

attempt to extend the results of Walsh [Wal09].

In particular, we show how the results of Walsh can be extended to veto elections of four or more candidates, but that this requires an adjustment to the algorithm for k-Way Partition. This confirms the statement by Walsh that all veto elections of a fixed number of candidates are potentially frequently easy to manipulate. We further show that while these adjustments can be used to solve manipulation of all scoring protocols, complexity results similar to that shown by Walsh for veto systems do not follow for all scoring protocols cases.

4.1 Manipulation as a Partition Problem

We show how to encode instances of the problem of manipulation in scoring protocols into instances of k-Way Permutation Partition, in order to solve manipulation using k-Way Permutation Partition algorithms. There are two relevant cases.

Theorem 4.1.1 *Consider an election of the scoring system $(\alpha_1, \ldots, \alpha_k)$ and candidates $c_1, \ldots, c_k$ with initial scores $s_1, \ldots, s_k$. Let the set of manipulative voters be V' of cardinality $||V'|| = m$ with weights $w_1, \ldots, w_m$.*

There exists a manipulation ensuring the victory of c_1 iff there exists a $(k-1)$-way permutation partition of the tuples $\{(s_2, \ldots, s_k)\} \cup \{(w_1\alpha_2, \ldots, w_1\alpha_k), \ldots, (w_m\alpha_2, \ldots, w_m\alpha_k)\}$ with maximum subset sum of $s_1 + \alpha_1 \sum_{1 \leq i \leq m} w_i$.

Proof: Without loss of generality, each voter from V' will choose c_1 as his or her first preference, giving c_1 a final score of $s_1 + \alpha_1 \sum_{1 \leq i \leq m} w_i$. To ensure the victory of c_1, the final score of any of the other candidates, $c_2, \ldots, c_k$, cannot exceed this quantity. As each voter of weight w_i is free to choose a permutation of scores $w_i\alpha_2, \ldots, w_i\alpha_k$ to give the candidates $c_2, \ldots, c_k$, this corresponds to the problem of $k-1$-Way Permutation Partition containing a vector containing these scores for each voter in V', as well as a vector representing the current scores of the $k-1$ nondistinguished candidates. The $k-1$ subsets in a permutation partition of these tuples, particularly their sum, thus represents the final score of the candidates $c_2, \ldots, c_k$.

A manipulation ensuring the victory of c_1 thus exists iff there is a $(k-1)$-way permutation partition of the tuples such that no subset exceeds the final score of c_1 mentioned earlier in sum. ■

It should also be noted that in this encoding, we are not solving an instance of k-Way Permutation Partition, but rather, finding at least one permutation partition such that the

sum of the maximum subset is above a certain threshold. We show how to implement this cutoff with pseudocode later in this chapter.

In some election systems, such as k-veto for k small, it is more intuitive and efficient to partition the vetoes we are distributing, as the tuples to partition would be given more zero elements. Recall that zero elements are dropped in the algorithms of SNP and RNP (recall from Section 2.4). We give such a reduction as follows.

Theorem 4.1.2 *Consider a q-veto election of candidates $c_1, \ldots, c_k$ with initial scores $s_1, \ldots, s_k$. Let the set of manipulative voters be V' of cardinality $||V'|| = m$ with weights $w_1, \ldots, w_m$. Without loss of generality, suppose $s_k \geq s_2, \ldots, s_{k-1}$.*

There exists a manipulation ensuring the victory of c_1 iff there exists a $(k-1)$-way permutation partition of the tuples $\{(s_k - s_2, \ldots, s_k - s_k)\} \cup \{(\underbrace{w_1, \ldots, w_1}_{q}, 0, \ldots, 0)\ , \ldots,$ $(\underbrace{w_m, \ldots, w_m}_{q}, 0, \ldots, 0)\}$ with minimum subset sum of at least $s_k - s_1$.

Proof: In this case, we are counting the total weight of the vetoes each nondistinguished candidates receives. Without loss of generality, no vetoes are given to the distinguished candidate. With some modification, this reduction also applies to cases of scoring protocols of the form $(\underbrace{\alpha, \ldots, \alpha}_{k-q}, \alpha_{k-q+1}, \ldots, \alpha_k)$ for $\alpha > \alpha_{k-q+1} \geq \ldots \geq \alpha_k$. This is particularly useful in cases where q is small relative to k. ■

4.2 Algorithms for k-Way Permutation Partition

In this section, we show how each of the known algorithms for k-Way Partition may be extended to that of k-Way Permutation Partition. Such algorithms allow us to empirically evaluate the complexity of manipulating scoring protocols.

Recall from Section 2.4 that two interesting partitioning heuristics for solving manipulation in scoring protocols include minimizing the maximum subset sum, and maximizing the minimum subset sum. We will consider the former for our algorithms in this section, as the latter can be implemented similarly by symmetry.

4.2.1 Extension of CKK to k-Way Permutation Partition

The CKK algorithm is the easiest to extend to k-Way Permutation Partition, as this algorithm uses tuples internally. We review the algorithm and give an extension of the CKK

algorithm to this problem as follows.

Consider a multi-set of k-tuples $S = \{x_1, \ldots, x_n\}$, where $x_i = (x_i^1, \ldots, x_i^k)$. We normalize the tuples so that $x_i^k = 0$ for each $1 \leq i \leq n$, and the tuples x_i are sorted in nonascending order by the first elements of each tuple. Each tuple is also sorted such that $x_i^1 \geq \cdots \geq x_i^k = 0$.

We consider the first two k-tuples, i.e., the two with the largest first elements. There are $k!$ possible combinations of permutations of the two k-tuples. We perform the search as a $k!$-ary search, with the heuristic of trying the combinations in the order of the greatest difference within the new tuple, i.e., the difference between the largest and smallest element in the tuple, in nondescending order. For example, to combine tuples $(5, 0, 0, 0)$ and $(4, 2, 0, 0)$, we permute them $(5, 0, 0, 0)$ and $(0, 4, 2, 0)$, arriving at the new tuple $(5, 4, 2, 0)$, with a greatest difference of 5. The next permutation we try would combine $(5, 0, 0, 0)$ and $(2, 4, 0, 0)$, arriving at $(7, 4, 0, 0)$ with a greatest difference of 7.

In each branch, we normalize the new tuple such that the last element, or the smallest element, of the tuple is 0, and that the tuples are sorted in nonascending order by their first elements.

We especially note that this algorithm is consistent with the CKK algorithm for 2-Way Partition and k-Way Partition given by Korf [Kor09], meaning that this algorithm will behave identically to the algorithms of Korf if it is given an instance of 2-Way Partition or k-Way Partition. This is a common property of most of the algorithms we will introduce in this chapter.

Subtree Pruning in *k*-Way Permutation Partition

As in the CKK algorithm for 2-Way Partition, we wish to prune the search as early as possible. In 2-Way Partition, this occurred when the largest element was greater than the sum of the remaining elements. We generalize this cut-off condition as follows.

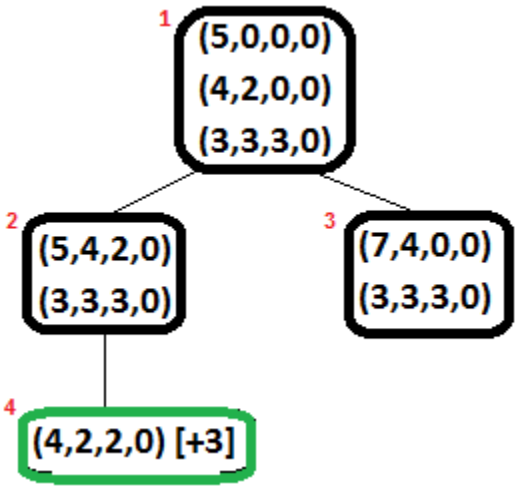
In k-Way Permutation Partition, we may cut off the search if we know that the first element (i.e., the element with the greatest difference among elements), cannot possibly be combined with the remaining elements to improve the best maximum subset sum found thus far, call it m.

Theorem 4.2.2 *A valid permutation partition for the tuples* $S = \{x_1 = (x_1^1, \ldots, x_1^k), \ldots, x_n = (x_n^1, \ldots, x_n^k)\}$ *such that no subset has a sum of more than* m *cannot exist if for some* $1 \leq r \leq k$,

$$\sum_{1 \leq i \leq r} x_1^i + \sum_{\substack{k-r+1 \leq i \leq k \\ 2 \leq j \leq n}} x_j^i > rm.$$

Proof: By the pigeonhole principle, at least one of the r sets that is given one of the elements $x_1^1, \ldots, x_1^r$ must exceed m in sum under these conditions, as they must receive a combined total exceeding rm in the final partition. This cut-off condition is consistent with that of CKK in 2-Way Partition, which is a special case of this condition for $k = 2$ and $r = 1$. ■

To illustrate this pruning technique, consider the following example, in which case we are partitioning the tuples $\{(5, 0, 0, 0), (4, 2, 0, 0), (3, 3, 3, 0)\}$. Part of the search tree given by the CKK algorithm on this instance is shown below. We represent, in square brackets, the amount subtracted from each element in a tuple during normalization. These numbers are to be added to the result of the permutation partition of the normalized instances.



In node 1, we initially combine tuples $(5, 0, 0, 0)$ and $(4, 2, 0, 0)$ into $(5, 4, 2, 0)$, and this new tuple with $(3, 3, 3, 0)$ to arrive at $(7, 5, 5, 3)$, which we normalize to $(4, 2, 2, 0)$. This results in a permutation partition with maximum sum of 7. It is also possible to combine $(5, 0, 0, 0)$ and $(4, 2, 0, 0)$ into $(7, 4, 0, 0)$. However, since $(7 + 4) + (3 + 0) = 14 \geq 2(6)$, it is impossible to obtain a partition with maximum sum of at most 6 from this branch, and it is thus pruned. This corresponds to the pruning condition above for $r = 2$. We will not combine 5 with 4, as this finds a partition with a maximum sum of at least 9, and the best partition currently has a maximum sum of 7.

Because the complete greedy and CKK algorithm for k-Way Partition is in general im-

practical for reasonably-sized integers when $k \geq 3$, a different form of algorithm was introduced in [Kor09]. Since k-Way Partition is a subclass of problems of k-Way Permutation Partition, this algorithm is equally impractical for this case in $k \geq 3$. We will thus evaluate how the algorithms introduced by Korf, namely SNP and RNP, can be extended to the problem of k-Way Permutation Partition.

4.2.3 Extension of SNP to k-Way Permutation Partition

We propose two natural extensions of the SNP algorithm introduced by Korf to that of k-Way Permutation Partition. In the first extension, we sequentially choose elements from each tuple, whereas in the second extension, we view the elements from each tuple individually. Both algorithms are consistent with the algorithm introduced by Korf. There are differing symmetry and pruning advantages to each algorithm, but as we will see later, the only reasonable extension of the RNP algorithm makes use of the latter method.

In SNP for k-Way Permutation Partition, instead of choosing one full subset, we must choose elements $y_i^1 \in \{x_i^1, \ldots, x_i^k\}$ for each $1 \leq i \leq n$ such that

$$\sum_{1 \leq i \leq n} y_i^1 \leq m,$$

where m is the maximum subset sum of the best partition found so far. For each such choice, we would then recursively find the best permutation partition of the $(k-1)$-element tuples containing the elements $\{x_i^1, \ldots, x_i^k\} \backslash \{y_i^1\}$ for each $1 \leq i \leq n$. As in the case of CKK above, we always normalize these tuples such that the smallest element of the tuple is zero in our recursive calls.

In the first extension of the SNP algorithm, we choose elements y_i^1 from the tuple x_i by depth-first search, choosing from tuples sequentially in nonascending order by their first element, as well as choosing the largest element among each tuple first. By starting with the largest elements, we increase the amount of pruning near the root of the tree. The reason for this is explained in more detail in [Kor09].

As in the case of SNP for k-Way Partition, We may also reduce the complexity of our search considerably by symmetry on the k chosen sets. We may assume without loss of generality that

$$\sum_{1 \leq i \leq n} y_i^1 \leq \lfloor \frac{t}{k} \rfloor,$$

where

$$t = \sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq k}} x_i^j,$$

as one such set must be the smallest of the k sets, and that for each successive set chosen the sum is no less than that of the previous set. By making this symmetry assumption, we may prune the search as soon as it becomes impossible to meet these summation constraints.

This algorithm is consistent with the SNP algorithm given by Korf, since given an instance of the special case of k-Way Partition, at each search node, given tuple x_i, the algorithm will consider x_i^1 and 0, which correspond to including or excluding this element. The pruning techniques also generalize those used in k-Way Partition in a consistent way.

As an example, consider an instance of 4-Way Permutation Partition of the tuples $\{(8,0,0,0),(6,2,0,0),(5,5,2,0),(5,5,0,0),(5,4,3,0)\}$, which has an optimal partition with a maximum subset sum of 13.

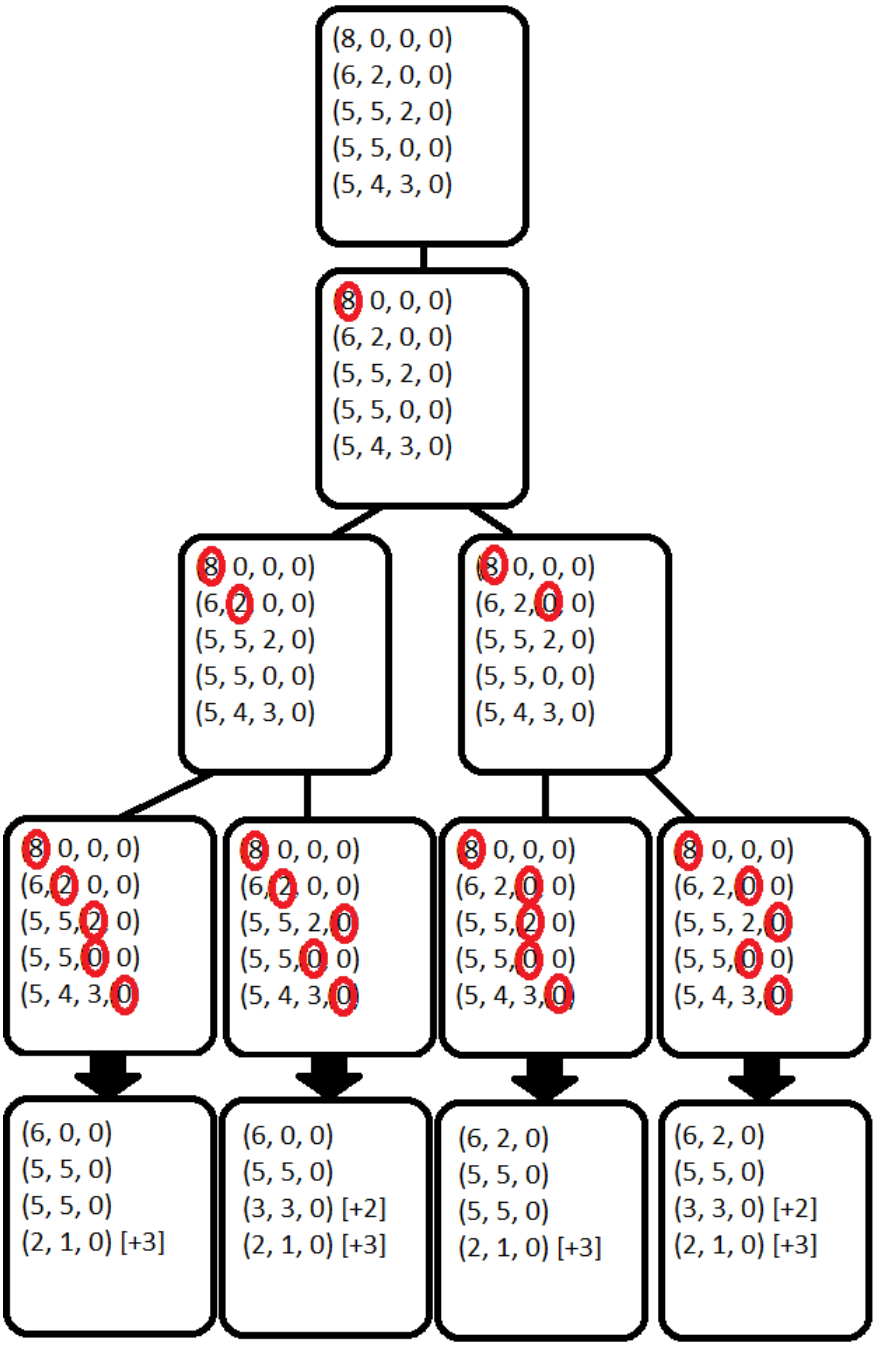
In this case, the sum of all elements is $t = 50$, and as such, we will choose a subset of sum at most 12. In the first branch of this top-level selection tree, we will choose element 8 from the first tuple, 2 from the second and third tuple, and 0 from the remaining tuples, giving us a 3-Way Permutation Partition instance of tuples $\{(6,0,0),(5,5,0),(5,5,0),(5,4,3)\}$. We normalize the tuple $(5,4,3)$ by subtracting 3 from each element, arriving at $(2,1,0)$, and adding 3 to the resulting best partition.

As it turns out, the best partition of this instance gives us a maximum subset sum of 14, which is suboptimal for the original instance. In the next branch, we would choose 0 from the third tuple, which also leads to a suboptimal partition. There are no other branches after choosing 8 and 2 from the first and second tuples respectively.

In the next branch, our only option after choosing 8 from the first tuple is to choose 0 from the second tuple. We could then choose 2 from the third tuple, which leaves us 0 as the only choice for the remaining tuples, as this subset cannot exceed 12 in sum. This also leads to a suboptimal partition. We thus now consider choosing 8, 0, and 0 from the first three tuples respectively. This forces the choice of 0 in the fourth tuple. Our first choice in the last tuple is 4, giving us an instance of 3-Way Permutation Partition of tuples $\{(6,2,0),(5,5,2),(5,5,0),(5,3,0)\}$. As it turns out, this selection results in a partition of maximum subset sum 13, which is optimal, since $\lceil \frac{50}{4} \rceil = 13$. We show a partial top-level search tree, with instances of 3-Way Permutation Partition, below.

In the instances of 3-Way Permutation Partition in the diagram below, we represent the amount subtracted from each element in a tuple in square brackets. Recall that in k-Way

Permutation Partition we normalize each recursive instance of $k-1$-Way Permutation Partition such that the $(k-1)^{st}$ element of each tuple is zero. These numbers are to be added to the final result of the best permutation partition on the normalized instance.



This example illustrates a major weakness of this algorithm, as it may be forced to consider insignificant elements, in this case those of weight 2, in tuples containing larger elements early in the search. Another significant weakness of this algorithm is that this concept cannot be easily extended to that of the RNP algorithm given by Korf, as the tuples are not considered sequentially. Nonetheless, it is a consistent extension of the SNP algorithm given by Korf, as zero elements chosen correspond to elements omitted in the first subset, and thus the selection of the first subset corresponds to the inclusion- exclusion search tree proposed by Korf.

As the RNP algorithm works with 2-way CKK at the top level, which is not applicable to that of permutations, an analogous extension of this algorithm to k-Way Permutation which views tuples sequentially does not follow. In the next section, we introduce another approach to k-Way Permutation Partition that gives rise to a different extension of both the SNP and RNP algorithms.

4.2.4 *k*-Way Permutation Partition as a Restricted *k*-Way Partition problem

In our second extension of SNP, as well as the extension for RNP, we make the observation that k-Way Permutation Partition is a restricted version of the k-Way Partition problem. More specifically, given tuples $S = \{x_1 = (x_1^1, \ldots, x_1^k), \ldots, x_n = (x_n^1, \ldots, x_n^k)\}$, we wish to find a k-Way Partition of the multiset of the union of all elements, $\{x_1^1, \ldots, x_1^k, \ldots, x_n^1, \ldots, x_n^k\}$, excluding zeros, with the additional constraint that each of the k subsets contains at most one element from each of the n tuples. We define this problem, k-Way Labeled Partition, as follows.

Name: k-Way Labeled Partition

Instance: A multi-set $S = (w_1, l_1), \ldots, (w_n, l_n)$ of pairs, each consisting of a weight and a label. Without loss of generality, $w_i \neq 0$ for all $1 \leq i \leq n$.

Question (decision): Are there disjoint and covering subsets $S = A_1 \cup \cdots \cup A_k$ such that each subset A_i contains elements of distinct labels and $\sum\{w|(w,l) \in A_1\} = \cdots = \sum\{w|(w,l) \in A_k\}$?

Question (optimization #1) Find disjoint and covering subsets $S = A_1 \cup \cdots \cup A_k$ where each subset A_i contains elements of distinct labels that minimizes $\max(\sum\{w|(w,l) \in A_1\}, \ldots, \sum\{w|(w,l) \in A_k\})$.

Question (optimization #2) Find disjoint and covering subsets $S = A_1 \cup \cdots \cup A_k$ where each subset A_i contains elements of distinct labels that maximizes $\min(\sum\{w|(w,l) \in A_1\}, \ldots, \sum\{w|(w,l) \in A_k\})$.

We give an example of 3-Way Labeled Partition as follows. Consider the multi-set $S = \{5_0, 5_1, 3_2, 2_2, 1_0, 1_1, 1_2\}$ (We notate the label as a subscript of the weight). One possible partition of this multi-set is $S = A_1 \cup A_2 \cup A_3$, where $A_1 = \{5_0, 2_2\}$, $A_2 = \{5_1, 1_0, 1_2\}$, and $A_3 = \{3_2, 1_1\}$. In this case, the subsets each have a total weight of at most 7, the optimum partition for this case.

Another restricted variation of interest for Partition is that of Balanced (2-Way) Partition [Yak96, MKAL03]. In this case, each of the two subset is to contain an equal cardinality of (unlabeled) elements.

As both the SNP and RNP algorithm work in a divide-and-conquer manner in which subpartitions are generated in sequence, the additional constraints of k-Way Labeled Partition can be implemented by additional restrictions in branching in each node of the search tree. This technique has also been applied to Balanced 2-Way Partition [ZMP11] in the context of the CKK algorithm. We give a brief description of how restrictions can be implemented to solve k-Way Labeled Partition, and thus, k-Way Permutation Partition.

In the SNP algorithm, we wish to choose a set of elements for our first subset. As in k-Way Partition, we consider the elements in nonascending order in an inclusion-exclusion search tree. In the SNP algorithm for k-Way Labeled Partition, we implement the restrictions in the top-level inclusion-exclusion search. More specifically, if an element with the same label has already been chosen, we must exclude this element from the set. On the other hand, if we are considering the last element of its label, and there are exactly k elements with this label, we must include it. By symmetry, if we choose to exclude an element, we should similarly exclude all subsequent elements that are similar (have the same weight and label). Finally, at each node, we consider the maximum sum we can choose for this subset from this point on. If this is less than the minimum we wish to choose for this subset, similarly to pruning techniques for SNP, we backtrack from the search node. Note that since zero elements are excluded, this algorithm is also consistent with the SNP algorithm given by Korf, as each element would have a distinct label. This algorithm behaves differently from our first extension of k-Way Permutation Partition in that the heavier elements of subsequent tuples are considered before lighter elements of earlier tuples.

We define pseudocode for the modified SNP algorithm as follows. In the function **SNP** below, we are given the set of labeled elements and the index of the element for which we

are contemplating inclusion. The parameter `min` gives a lower bound for the sum of the first subset to be chosen, while `prune` tells us how good the partition to be found should be. The variables `sum`, S', and C tell us, respectively, the current sum of the subset to be chosen, the elements not chosen which are to be recursively partitioned into $k-1$ subsets, and the labels of the elements that have been chosen.

SNP($S = \{(w_1, \ell_1), \ldots, (w_n, \ell_n)\}$,$k$,
`min`, `prune`,
i, `sum`, S', C)

if $(i > n)$
return **SNP**($S', k-1$, `sum`, `prune`, $0, 0, \emptyset, \emptyset$)

if $(\ell_i \notin C)$ **and** $(\texttt{sum} + w_i \leq \texttt{prune})$
$\texttt{best}_1 \leftarrow$ **SNP**(S, k, `min`, `prune`, $i+1$, `sum` $+ w_i, S', C \cup \{\ell_i\}$
if $(\texttt{best}_1 \leq \texttt{prune})$ **return** `prune`

if $(||\{((w, \ell) \in S | \ell = \ell_i\}|| \leq k-1)$ **and**
$(\texttt{sum} + \sum_{l \notin C} \max\{w | (w, \ell) \in \{(w_{i+1}, \ell_{i+1}), \ldots, (w_n, \ell_n)\} \geq \texttt{min})$
$\texttt{best}_2 \leftarrow$ **SNP**(S, k, `min`, `prune`, $i+1$, `sum`, $S' \cup \{(w_i, \ell_i)\}, C$)
if $(\texttt{best}_2 < \texttt{prune})$ **return** `prune`

return $\min(\texttt{best}_1, \texttt{best}_2)$

In the pseudocode above, we recursively compute the best partition in which the first subset includes the element (w_i, ℓ_i). The maximum subset sum is stored in $\texttt{best}_1$. The element (w_i, ℓ_i) can be chosen if no elements of the same label have been chosen and the current sum, including this element, will not exclude `prune`, our target.

Similarly for $\texttt{best}_2$, we compute the best partition in which the first subset does not include the element (w_i, ℓ_i). We can exclude this element in the first subset if there are no more than $k-1$ elements of this label remaining and also the sum of the first subset, if the largest remaining elements are chosen for each label, will be at least `min`. If in either case we find a partition in which the maximum subset sum is at most `prune`, we may prune the search as a good enough partition has been found. In the event that neither partition beats this

constraint, we return the maximum subset sum of the better partition, $\min(\texttt{best}_1, \texttt{best}_2)$.

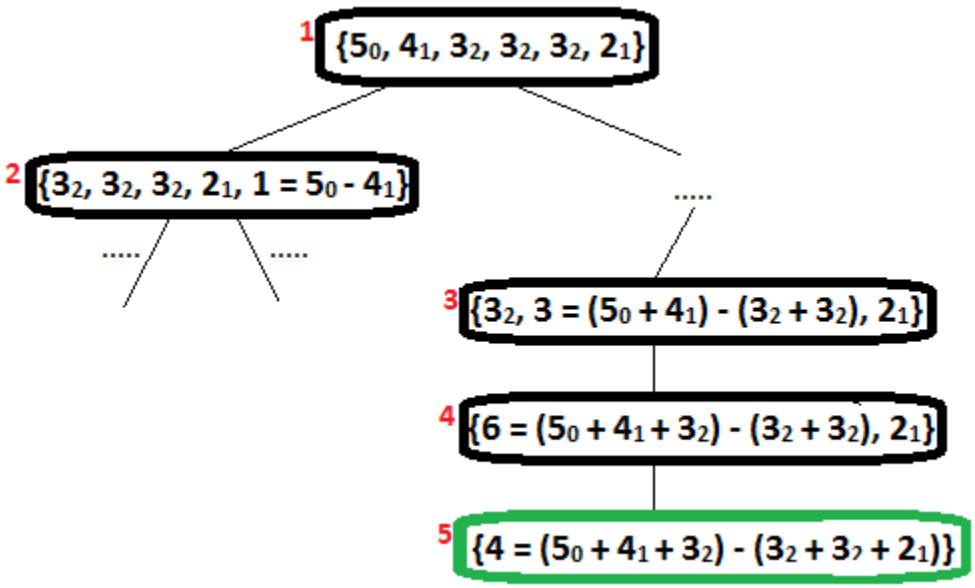
In the base case, in which we have included or excluded all n elements of S, we recursively partition the remaining elements, S', into $k-1$ subsets, using the sum of the first subset as the minimum, min. Recall that we are assuming that the subsets will be chosen in nondescending order to break symmetry. Initially, the minimum subset sum variable, `min`, is set to $\frac{\sum\{w|(w,\ell) \in S\}}{k}$, as the first subset sum will be at most that of the other subsets.

This extension of the SNP algorithm for k-Way Permutation Partition has the advantage of having a similarly well-defined extension of the RNP algorithm proposed by Korf. In our extension of RNP on k-Way Labeled Partition, we perform the CKK algorithm on the labeled elements, under the constraint that each of the two subsets associated with each partial partition receive no more than $\frac{k}{2}$ elements of each label. As each of the two branches in a CKK search node entail combining two subpartitions, we achieve this by refraining from combining subpartitions in which one or both subsets exceeds $\frac{k}{2}$ elements of any label.

We consider the same example as above, in which we wish to partition the tuples $\{(8,0,0,0),(6,2,0,0),(5,5,2,0),(5,5,0,0),(5,4,3,0)\}$. To reduce this instance of k-Way Permutation Partition to an instance of k-Way Labeled Partition, we sort the union of the elements of each tuple, excluding zero, arriving at the labeled multiset $\{8_0, 6_1, 5_2, 5_2, 5_3, 5_3, 5_4, 4_4, 3_4, 2_1, 2_2\}$. Each element is labeled with the tuple it originated from.

The first branch of the top-level CKK search combines elements 8_0 and 6_1, adding a new element of net weight 2 to this set. Since we are interested in the further partitioning of the 2-Way Partition of this set, we must keep track of how we arrive at our net elements. We could represent this resulting multiset as follows: $\{5_2, 5_2, 5_3, 5_3, 5_4, 4_4, 3_4, 2 = 8_0 - 6_1, 2_1, 2_2\}$. Skipping a few steps, eventually this top-level CKK algorithm produces a partition of sets $\{6_1, 5_2, 5_3, 4_4, 3_4, 2_2\}$ and $\{8_0, 5_2, 5_3, 5_4, 2_1\}$ at its first left-hand-side leaf. This corresponds to two instances of 2-Way Labeled Partition, which can be solved using the CKK algorithm. In this case, this first leaf will also yield the best partition of maximum subset sum 13, but possibly significantly faster than the SNP extension, as CKK is notably fast relative to the more complex inclusion-exclusion search trees of SNP.

To illustrate the effects of the additional constraints on Korf's RNP algorithm, we give another example. In this case we wish to partition the tuples $\{(5,0,0,0),(4,2,0,0),(3,3,3,0)\}$, giving the following instance of 4-Way Labeled Partition: $\{5_0, 4_1, 3_2, 3_2, 3_2, 2_1\}$. We show, partially, the top-level CKK search tree.



In the search tree shown above, the left-hand subtree rooted at node 2 yields a best partition with a maximum subset sum of 7. Consider node 3 in the diagram above. We do not take the difference between 3_2 and $3 = (5_0+4_1)-(3_2+3_2)$, because this results in the left-hand subset containing three elements of label 2. In node 4, taking the difference between the two elements results in the partition $\{5_0, 4_1, 3_2\} \cup \{3_2, 3_2, 2_1\}$, which we recursively partition into two subsets each. However, we do not take the sum of the elements $6 = (5_0+4_1+3_2)-(3_2+3_2)$ and 2_1, because this results in the left-hand subset containing elements of total weight 14, rendering it impossible to beat the current maximum subset sum of 7.

In the pseudocode given below, we are evaluating k-way partitions of the elements in S. Our goal is to minimize the maximum sum of the subsets. The other interesting optimization of maximizing the minimum sum of the subsets can be implemented similarly by symmetry.

There are five inputs to the $\mathbf{RNP}_{\mathbf{inner}}$ function. The first input is the set of weights, S, in the current branch of the top-level CKK division. Each weight is further labeled with the subpartition of elements such that the weight is given by the difference of the subpartition. An example of this labeling is shown in the example search tree above: For instance, in node 3 of the search tree, the second weight of 3 is labeled by the partition $5_0, 4_1 \cup 3_2, 3_2$, which has a difference of 3.

The input min tells us the minimum that each subset should total, due to the pruning techniques used in the algorithm. `best` tells us the best result found so far, i.e., the lowest maximum subset total we have attained, while `prune` tells us the best result we will need.

This could be due to pruning techniques, or as an application of the algorithm for the problem we are interested in solving, and allows the algorithm to possibly terminate early. For example, in most cases of manipulation, it suffices to find a partition better than some threshold; it is not required to find the best partition.

The function $\texttt{RNP}_{\texttt{inner}}$ recursively works on the initial two-way partitioning of the elements, keeping track of the elements making up each difference, and calls the top-level function RNP to evaluate $\frac{k}{2}$-way partitions at the leaves of the CKK search tree.

$\mathbf{RNP_{inner}}(S = \{x_1 = \sum_{(w,l)\in A_1} w - \sum_{(w,l)\in B_1} w,$
$x_2 = \sum_{(w,l)\in A_2} w - \sum_{(w,l)\in B_2} w, \ldots,$
$x_n = \sum_{(w,l)\in A_n} w - \sum_{(w,l)\in B_n} w\},$
$k,$
$\min, \texttt{best}, \texttt{prune})$

if $(n = 1)$; Base case, we are done with the top-level CKK division
$\texttt{best}_1 \leftarrow \texttt{RNP}(A_1, \frac{k}{2}, \min, \texttt{best}, \texttt{prune})$; Recursively partition A_1 $\frac{k}{2}$ ways.
if $(\texttt{best}_1 \geq \texttt{best})$ **return** best. ; Can't beat the best partition found.
if $(\texttt{best}_1 > \texttt{prune})$ $\texttt{prune} \leftarrow \texttt{best}_1$. ; Don't need better partition in B_1.
$\texttt{best}_2 \leftarrow \texttt{RNP}(B_1, \frac{k}{2}, \min, \texttt{best}, \texttt{prune})$; Recursively partition B_1 $\frac{k}{2}$ ways.
if $(\texttt{best}_2 \geq \texttt{best})$ **return** best. ; Did not beat the best partition.
return $\max(\texttt{best}_1, \texttt{best}_2)$; Return overall result of this partition

else
if each of $A_1 \cup B_2$ and $B_1 \cup A_2$ sum to at most $k(\texttt{best} - 1)$ and contain at most $\frac{k}{2}$ elements from each tuple ; Try difference if possible
$\texttt{best}_1 \leftarrow \texttt{RNP}_{\texttt{inner}}(\{(x_1 - x_2) = \sum_{(w,l)\in A_1\cup B_2} w - \sum_{(w,l)\in A_2\cup B_1} w, \ldots,$
$x_n = \sum_{(w,l)\in A_n} w - \sum_{(w,l)\in B_n} w\},$
$k, \min, \texttt{best}, \texttt{prune})$
if $(\texttt{best}_1 \leq \texttt{prune})$ **return** prune ; Don't need better partition
if $(\texttt{best}_1 < \texttt{best})\texttt{best} \leftarrow \texttt{best}_1$; Update best partition found

if each of $A_1 \cup A_2$ and $B_1 \cup B_2$ sum to at most $k(\texttt{best} - 1)$ and contain

at most $\frac{k}{2}$ elements from each tuple ; Try sum if possible

$\texttt{best}_2 \leftarrow \texttt{RNP}_{\texttt{inner}}(\{(x_1 + x_2) = \sum_{(w,l)\in A_1 \cup B_1} w - \sum_{(w,l)\in A_2 \cup B_2} w, \ldots,$
$x_n = \sum_{(w,l)\in A_n} w - \sum_{(w,l)\in B_n} w\},$
$k, \min, \texttt{best}, \texttt{prune})$

if $(\texttt{best}_2 \leq \texttt{prune})$ **return** `prune`

if $(\texttt{best}_2 < \texttt{best})\texttt{best} \leftarrow \texttt{best}_2$

return `best`

At the top level, we start with the individual elements. We prune the search additionally by using the KK heuristic on the corresponding k-Way Permutation Partition instance, possibly improving the outcome of the best partition thus far, i.e., the input `best`.

RNP$(S = \{(w_1, l_1), \ldots, (w_n, l_n)\}, k, \min, \texttt{best}, \texttt{prune})$

`kk` $\leftarrow$ Karmarkar-Karp heuristic in corresponding k-Way Permutation Partition instance of S.

if $(\texttt{kk} \leq \texttt{prune})$ **return** `prune` ; Don't need better partition

if $(\texttt{kk} < \texttt{best})$ `best` $\leftarrow$ `kk` ; Update best partition found

return $\textbf{RNP}_{\textbf{inner}}(S, k, \min, \texttt{best}, \texttt{prune})$

The main difference between the implementation of RNP for this restricted partition problem and that of Korf's implementation for k-Way Partition is the extra restriction for combining weights, as the subsets involved must not exceed $\frac{k}{2}$ elements from any tuple. The SNP algorithm can be similarly implemented.

4.3 Experimental Results

We test the feasibility of using the algorithms for k-Way Permutation Partition described in this chapter for solving manipulation problems of various election systems using the connections in the previous section. To eliminate the issues of computer architecture and focus on the algorithm computationally, in each case, as a benchmark, we count the number of branches evaluated in invoking the algorithms in question, as opposed to runtime, for cases

of interest. We also evaluate the standard error to provide 95% confidence intervals on the experimental data, to ensure statistical significance.

In all of the tests below, we choose voter weights uniformly from the range $[1, 2^n]$, where n is the number of manipulators. The reason for this choice of range is from the observation of the phase transition in Partition given in [GW96]. It is observed that when partitioning n uniformly random integers in the range $[1, 2^m]$, the problem exhibits a phase transition when the constrainedness parameter $\frac{n}{m}$ approached a constant of roughly 1. This choice of range for voter weights also preserves the NP-completeness of the NP-complete manipulation problems we will look at.

It should be noted that in [Wal09] that the voter weights are uniformly chosen from a fixed range of $[1, 256]$. The resulting problem of manipulation under this condition is polynomial-time computable, due to dynamic programming over the weights. The empirical results are, however, unlikely to be unaffected by this theoretical result, as the polynomial exponent for such a theoretical algorithm are significant, especially in relationship to the size of the problem instance.

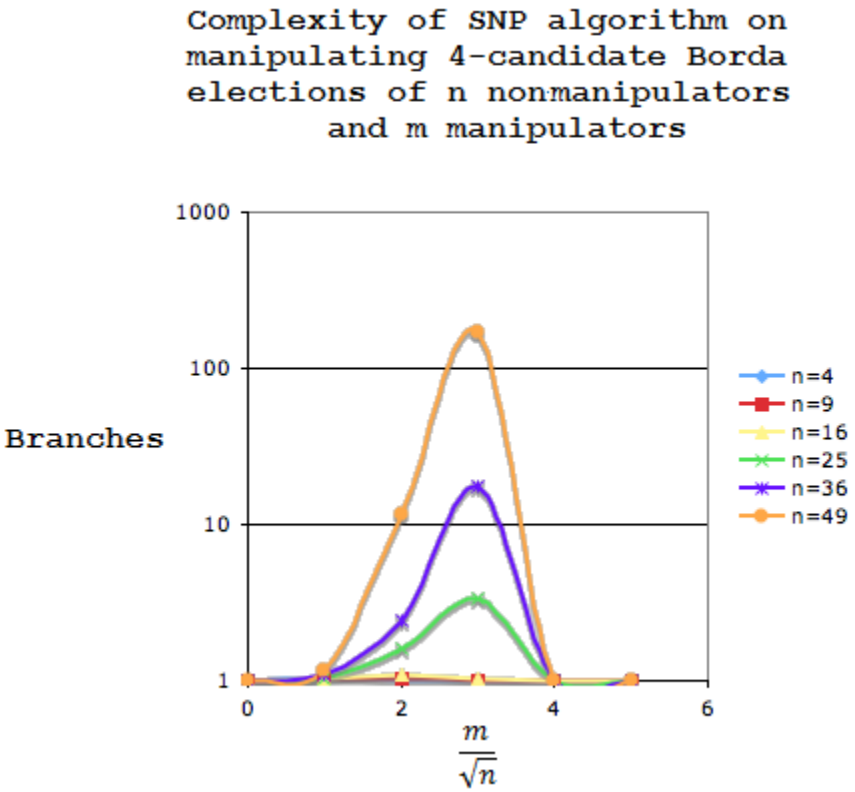
The preference orderings of each voter are uniformly and independently chosen as a random permutation. While this is an unnatural distribution of voter preferences in practical settings, this distribution allows us to gain an understanding as to the frequency of instances that evade the solution of Partition algorithms, and give us an understanding on where the hard instances of manipulation lie.

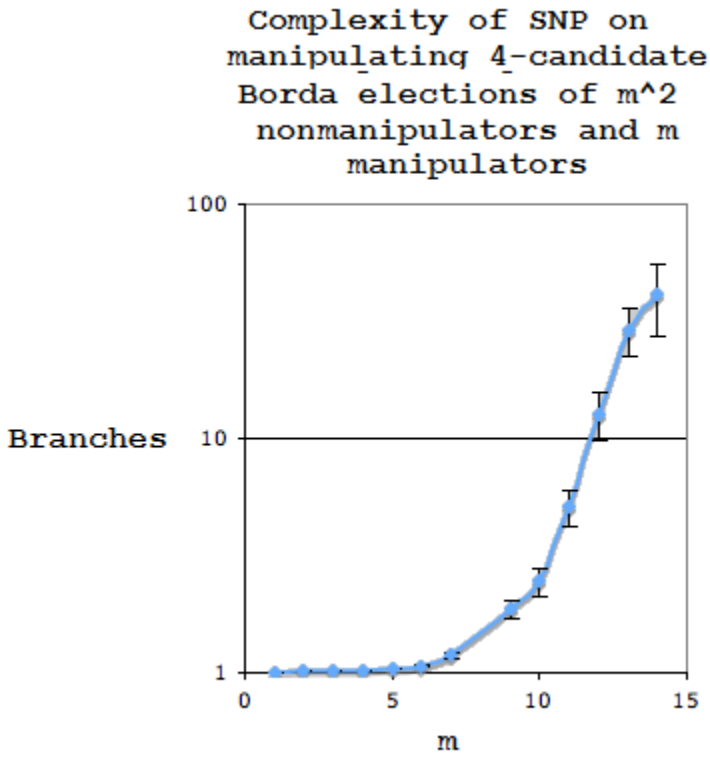
Our first test is for the case of manipulating weighted 2-approval and veto elections of a fixed number, k, of candidates, as the $(k-1)$-Way Permutation Problem instances for these cases are relatively similar to that of $(k-1)$-Way Partition. We found that, for these simple cases, the runtime is similar to that of the 3-candidate cases tested by Walsh, that most cases can be solved in an average of about one branch, and that worst-case hardness stems from an exponentially rare set of hard instances. Most interestingly, hard instances are not directly related to the phase transition of this problem. However, these results require the use of $(k-1)$-Way Permutation Partition, due to the arbitrary nature of the initial scores. This result also holds for scoring protocols of the form $(\alpha_1, \alpha_2, 0, \ldots, 0)$ for $\alpha_1 > \alpha_2$. In both this case and that of 2-approval and veto, all but one of the tuples have only one nonzero element.

We test the case of 4-candidate Borda elections, which does not have the form mentioned above. As with all of the plots in this section, we plot the 95% confidence interval of the mean of our testing. As demonstrated in [Wal09] for elections of three candidates, the phase

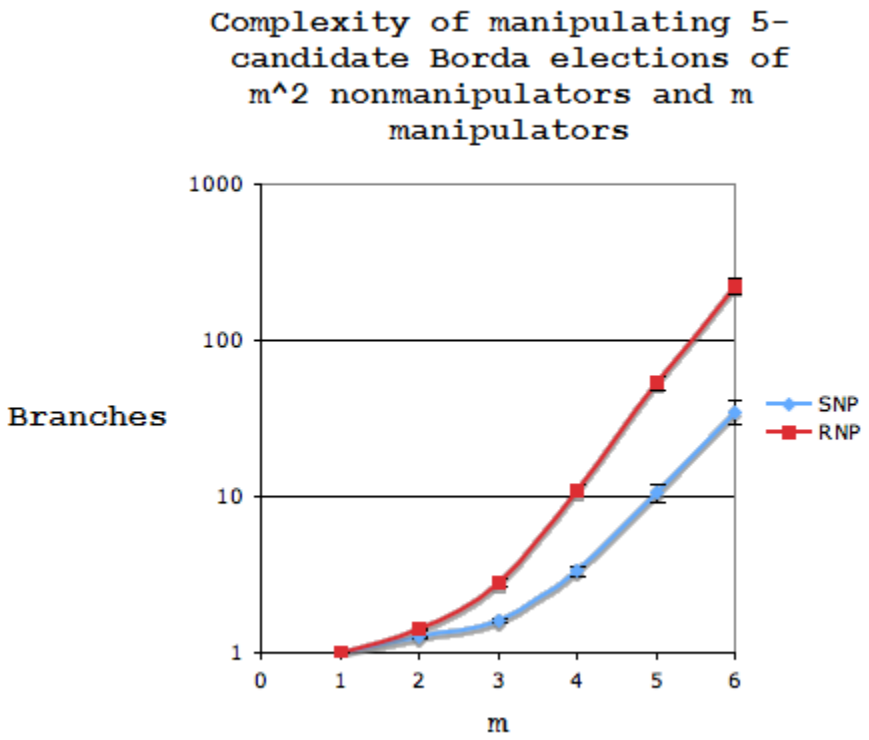
transition of this problem also occurs for $m = cn^2$ for some constant $c > 0$.

In the first chart, we scale the number of manipulators, m, across the phase transition of this problem for different values of n, the number of nonmanipulators. Note the increasing peaks in complexity near the phase transition as n is increased. In the second chart, we plot the runtime of invoking the SNP algorithm for an instances of manipulating 4-candidate Borda elections of $m = n^2$ nonmanipulative voters and n manipulators, near the phase transition of this problem.



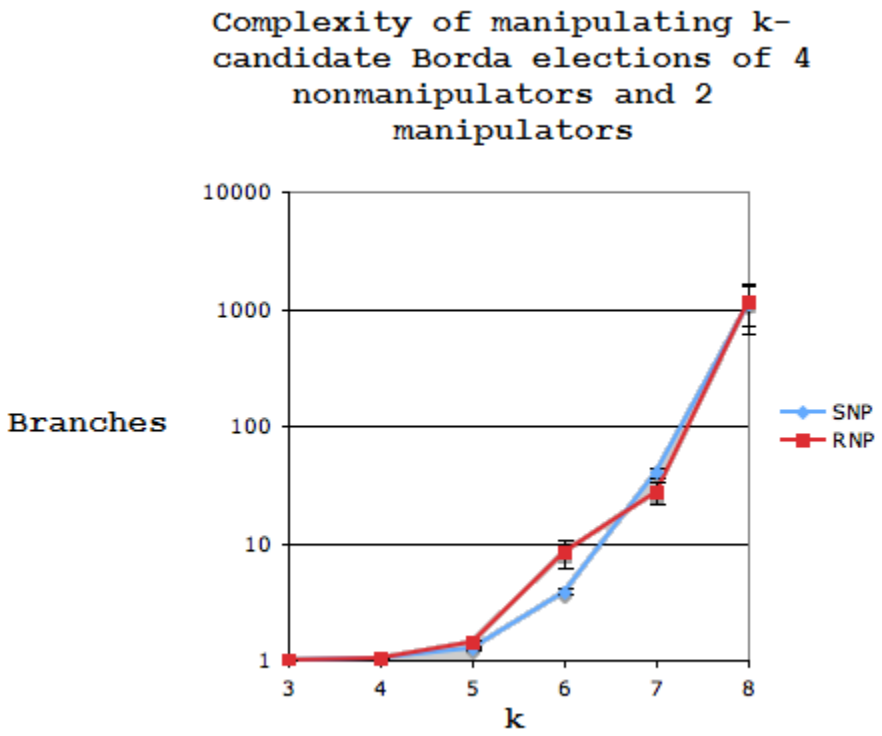


As we can see in this plots, the runtime of evaluating uniformly random instances of manipulating weighted 4-candidate Borda elections soars exponentially for instances even remotely near the phase transition of this problem. This result is in direct contrast to the results in Walsh for 3-candidate elections. Our next test case involves 5-candidate Borda elections, for which we have a choice between the SNP and RNP algorithms. We observe that the RNP algorithm is significantly slower than that of SNP, contrary to the results in Korf in the general setting. Neither algorithms extend the results of Walsh. These exponential empirical results are common for all scoring protocols except for 2-approval and veto, including k-approval for $k \geq 3$ and k-veto for $k \geq 2$.

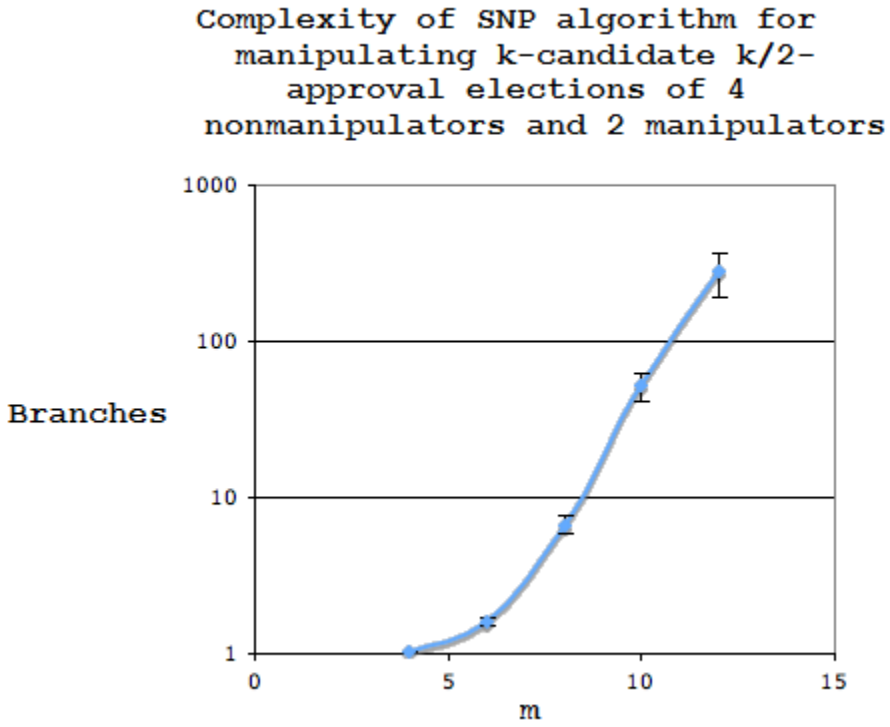


Our next two tests involves scaling the number of candidates in the election. We wish to investigate how the runtime complexity of the algorithms in question scale with the number of candidates in the election.

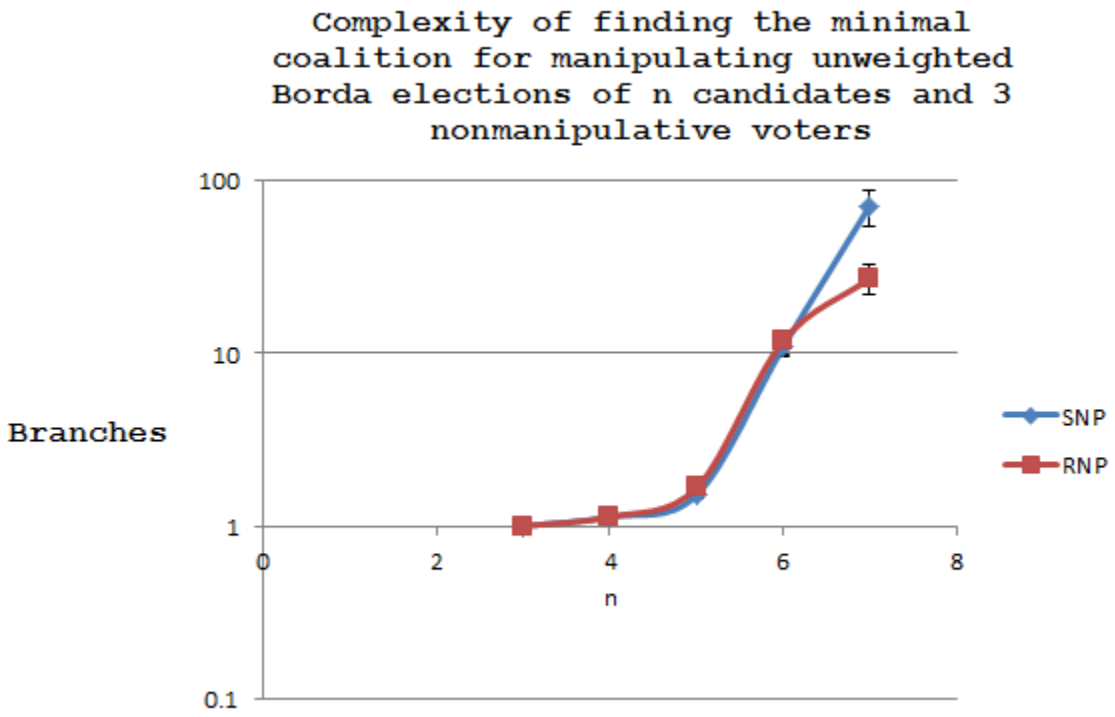
Below, we are plotting the runtimes of our algorithms on the case of manipulating k-candidate Borda elections for a fixed number of nonmanipulators and manipulators, 4 and 2 respectively. The exponential results demonstrate that the SNP and RNP algorithms do not perform well asymptotically when the size of the candidate set is increased, even for a small voter set.



In the next case, we test a simpler family of scoring protocols, that of k-candidate $\frac{k}{2}$-approval elections, using the SNP algorithm (the runtime of the RNP algorithm is similar). This case is of interest as it is the most complex approval case for our algorithms, as one may choose to partition approvals or vetoes. The exponential nature of these results also show that the runtime of these algorithms stem from the number of candidates, as opposed to the complexity within the scoring protocol itself.



In our last test, we evaluate the effect of invoking our algorithm for an NP-complete unweighted manipulation problem, the manipulation of the Borda family of scoring protocols. In [ZPR09], a greedy algorithm approximating the minimum coalition size for manipulating unweighted Borda election with an additive error of at most one is given. Using this greedy algorithm in conjunction with our partition algorithms, we find the smallest number of unweighted nonmanipulators needed to elect the preferred candidate, and note the search tree complexity on the following diagram.



These results further confirm that the complexity of these algorithms stem from the candidate set size. It is interesting to note that in this case, RNP is indeed faster than SNP, as was in the cases of k-Way Partition evaluated by Korf.

4.4 Discussion

These new algorithms show how the relationship between k-Way Partition and manipulating veto elections can be extended to that of manipulating other scoring protocols, a problem seemingly unrelated to k-Way Partition. These new findings allow one to develop a better understanding of the inner workings of these manipulation problems. Known algorithms for partition-type problems, including Sequential Number Partitioning (SNP) and Recursive Number Partitioning (RNP), have the property of finding a good partition relatively quickly when one exists, and this property appears to extend to that of our new problems. Although RNP has been demonstrated to be faster than SNP for the special case of k-Way Partition, this simplification does not appear to extend more generally here, particularly for more

complex systems such as Borda. Using these new problems, we confirm the statement by Walsh that the manipulation of veto candidates of more than three candidates is often easy, but this requires significant modification of the k-Way Partition problem.

In each case, the best-known algorithms of Korf, and very natural extensions and generalizations thereof, do not extend the conclusions of Walsh for elections of more than three candidates other than veto, but instead show that scoring protocols in general are not directly vulnerable to manipulation via the known algorithms for Partition. However, this may be due to either the strength of elections having more than three candidates, or weakness of the best-known algorithms for k-Way Partitioning and the extensions given in this chapter. We leave this as an open problem.

We note that there are also inherent weaknesses of the known algorithms for k-Way Partition. The obvious weakness of RNP, for instance, is that it can only be applied to cases of k-Way Permutation Partition for k even. This weakness is demonstrated clearly in Korf, in which it is shown that 3-Way Partition is almost as slow as 4-Way Partition, and by extension, similar results hold for k-Way Permutation Partition. In the cases where RNP would otherwise be more efficient than SNP, it is not available as an option for k odd as the CKK algorithm used at the top-level division is designed to enumerate partitions which are very close to being even. We leave possible improvements to these issues as an open problem.

A second weakness of RNP relates to the top-level CKK division, which divides the initial set of labeled elements into two. Recall that in the CKK algorithm, the algorithm tracks a list of 2-way partitions of subsets of the original set. If either subset in any of the 2-way partitions in such a list cannot be furthered partitioned $\frac{k}{2}$ ways in a way better than the best partition found so far, it is clearly fruitless to continue on this search branch, as the final 2-way partition cannot yield a better overall k-way partition. One way to capitalize on this insight is to perform the algorithm recursively on the instances of $\frac{k}{2}$-Way Permutation Partition at the search nodes of interest. Unfortunately, this will only prune the top-level search tree only if a better partition does not exist, and may be expensive, as the number of such nodes, absent pruning, may be exponential. A possible open problem of interest is thus a study of the tradeoffs in making such prunings, determining fast heuristics of when to perform this test within the tree.

Resolving these two weaknesses of the RNP algorithm can help explain the anomaly of why despite being faster than SNP in k-Way Partition [Kor09, Kor10], it behaved unusually slow in general for k-Way Permutation Partition.

Another problem of interest involves using variations of these new algorithms for polynomial-time computable approximation algorithms, a problem studied in the context of different election systems in [BFH+08]. Since it is also known that the CKK algorithm, and possibly the extensions thereof, are especially fast at finding a good partition when one exists, approximation and probabilistic algorithms may exist to capitalize on the tradeoffs in performing an incomplete search. Interesting optimization and approximation functions may include minimizing the number of manipulators needed, or relaxing the restrictions of the manipulation problem. The problem Partition has a fully polynomial-time approximation scheme (FPTAS), which may also be extended to problems like k-Way Permutation Partition. It would be of interest to evaluate the tradeoffs encountered using such an algorithm in the context of manipulating scoring protocols.

As SAT is a much better studied problem, and much more versatile for solving other problems in voting systems, in the next chapter we will consider the application of using SAT solvers to solve some manipulation problems. We show how SAT solvers can be applied to other problems of interest besides manipulation, such as bribery and control by adding or deleting candidates, and in Chapter 6, we show how SAT solvers can also be applied to the problem of finding the winner in some elections in which it is hard to, such as Dodgson and Young elections.

Chapter 5

SAT Solver Approaches to Problems in Voting

In [Con10], a reduction from manipulation in some election systems to 0-1 Integer Linear Programming, and in turn into SAT, is given. 0-1 Integer Linear Programming is a versatile problem that can be used to encode instances of many interesting problems, including the Traveling Salesman Problem [MTZ60] and Machine Learning [Wag59]. Because 0-1 ILP can be seen as a generalization of SAT on formulas of conjunctive-normal form, solutions of both problems share similarities [LZD04].

Connett used the reduction as a means of determining the minimal coalition required to manipulate elections. In doing so, the phase transition of manipulation is demonstrated empirically, in that $\Theta(\sqrt{n})$ manipulators are required to influence an election of n nonmanipulators in a number of interesting election systems, including several scoring protocols and some elections involving multiple rounds. The exact constant factor in this asymptotic observation is different for each election system. In this chapter, we evaluate the complexity of invoking state-of-the-art SAT solvers [GKS10] on SAT instances resulting from the reduction given by Connett and other natural reductions from manipulation problems to SAT.

We further extend the reductions given by Connett to the problems of bribery and control, reducing them to instances of 0-1 Integer Linear Programming, and in turn into instances of SAT using the methodologies introduced in [War98] and used by Connett [Con10].

We observe that the reduction given by Connett may be improved by breaking symmetry. In particular, the reduction allows many similar manipulations, such as those with unweighted manipulators transposed or with similar preference orderings, to be represented by multiple instantiations of the variables of the problem. We show how symmetry breaking

can improve the asymptotic complexity of invoking SAT solvers on these instances. Symmetry breaking is also evaluated more generally for the problem of SAT in [ARMS02b].

5.1 Manipulation as a 0-1 Program

0-1 Integer Linear Programming can be used to model both weighted and unweighted manipulation problems. A general reduction for many systems of interest, both weighted and unweighted, is given in [Con10]. In the reductions given by Connett, the preference of each voter between each pair of candidates is given by a variable. Because the preferences of the voters in scoring protocols are transitive, the reduction used, and in particular the variables used to represent the individual votes, can be simplified, as pairwise preferences for each voter are not needed, and the voter preferences can be represented as a linear list. This representation reduces "thrashing" by the SAT algorithm, and also eliminates the constraints needed to enforce transitivity. For example, in the representation of Connett, the algorithm can thrash by pursuing a search tree branch in which the preferences assigned are intransitive to this point. We summarize the Connett reduction with this simplification as follows. In particular, we show how transitive preferences can be encoded as a linear list as opposed to a table of pairwise preferences.

Theorem 5.1.1 *Consider the problem of weighted manipulation in a scoring protocol of m candidates, $(\alpha_1, \ldots, \alpha_m)$, in which there are n established voters and n' manipulators. This problem can be formulated as a* 0-1 *Integer Linear Program consisting of $\Theta(m^2 n')$ variables and $\Theta(mn')$ linear constraints.*

Proof: Consider an election of m candidates, $C = \{c_1, \ldots, c_m\}$. Let c_1 be the distinguished candidate whom we wish to ensure victory for, and for $c \in C$, let $s(c)$ be the initial score of candidate c. Let there be n' unestablished voters, $V' = \{v'_1, \ldots, v'_{n'}\}$.

Without loss of generality, each unestablished voter ranks c_1 as their first preference. For each $1 \leq i \leq n'$, $2 \leq j \leq m$, and $2 \leq \ell \leq m$, let $x_{i,j,\ell} = 1$ iff voter v'_i ranks candidate c_j in its ℓ^{th} position. Each voter ranks each candidate exactly once, and each voter gives each ranking position exactly once, corresponding to the linear constraints $\sum_j x_{i,j,\ell} = 1$ for all i, ℓ, and $\sum_\ell x_{i,j,\ell} = 1$ for all i, j. For each $2 \leq j \leq m$, candidate c_j finishes with score $s(c_j) + \sum_{i,\ell} w(v'_i)\alpha_\ell x_{i,j,\ell}$, which must be at most the final score of c_1, which is $s(c_1) + \sum_i w(v'_i)\alpha_1$. ■

A major problem with such a reduction is that it may still be inefficient for cases of unweighted voters or simple scoring protocols in which symmetry exists. Symmetry, in which similar reductions can be reached by more than one instantiation of the variables in the encoding, is a weakness because a depth-first search solver for SAT may only declare an instance to be unsatisfiability after an exhaustive search which clears each branch.

In the reduction of Connett [Con10], in cases such as in problems of unweighted voters and simple scoring protocols such as k-approval, the same manipulation may be represented by many instantiations of the variables. For example, in an unweighted manipulation problem, the identities of the voters giving each preference may be permuted to give similar manipulations. Many simple scoring protocols, such as k-approval, have identical scores within the vector for some positions, allowing parts of each voter's preferences in a manipulation to be permuted. This requires the algorithm to repeat similar search branches within the search tree, all of which may lead to unsuccessful manipulations. The amount of repetition in the search tree grows exponentially with respect to the cardinalities of the candidate and voter sets.

We give two cases where symmetry breaking may help us reduce the search tree complexity of solving our reductions to 0-1 ILP: that of weighted k-approval elections and unweighted scoring protocols.

In the case of k-approval elections of m candidates, the approvals given by each candidate are not ordered, and thus it suffices to choose k candidates to approve. In all unweighted cases, the voters are interchangeable, and only the total number of voters giving a certain preference is of importance. Because of the exponential nature of permutations, we suspect that reducing symmetry can have a profound effect especially on the case of families of scoring protocols. This is because the improvements would be more significant as the number of permutations of each possible manipulation is increased. We show how to break these symmetries in the following reductions.

Theorem 5.1.2 *Consider the problem of weighted manipulation in a k-approval election of m candidates, in which there are n established voters and n' manipulators. This problem can be formulated as a* 0-1 *Integer Linear Program consisting of $\Theta(mn')$ variables and $\Theta(m+n')$ linear constraints.*

Proof: Consider a k-approval election of m candidates, $C = \{c_1, \ldots, c_m\}$. Let c_1 be the distinguished candidate for whom we wish to ensure victory, and for $c \in C$, let $s(c)$ be the initial score of candidate c. Let there be n' unestablished voters, $V' = \{v'_1, \ldots, v'_{n'}\}$.

Without loss of generality, each unestablished voter approves c_1. For $1 \leq i \leq n'$ and $2 \leq j \leq m$, we indicate whether or not v'_i approves c_j as the variable $x_{i,j}$. Each voter v'_i must approve exactly $k-1$ candidates other than c_1, and thus $\sum_j x_{i,j} = k-1$. For each $2 \leq j \leq m$, we can also ensure that c_1 beats or ties each other candidate c_j using the inequality $s(c_j) + \sum_i w(v'_i)x_{i,j} \leq s(c_1) + \sum_i w(v'_i)$. ∎

In the above reduction, we avoid giving the exact rankings among candidates a particular voter will approve or disapprove, as all approvals are equally weighted. In an unweighted election, the identities of the voters giving the rankings to each candidate are not important; only the cardinality of voters giving each candidate each ranking are. We show how to break this symmetry in the following reduction for manipulating unweighted scoring protocols.

Theorem 5.1.3 *Manipulation of unweighted m-candidate scoring protocols, in which there are n' manipulators can be formulated as a* 0-1 *Integer Linear Programming instance consisting of $\Theta(m^2n')$ variables and $\Theta(m^2)$ constraints.*

Proof: Consider an unweighted election of candidates $C = \{c_1, \ldots, c_m\}$ under the scoring protocol $(\alpha_1, \ldots, \alpha_m)$, and let c_1 be the distinguished candidate. Let there be n' unestablished voters, $V' = \{v'_1, \ldots, v'_{n'}\}$.

Without loss of generality, each unestablished voter ranks c_1 as their favorite choice. For each $0 \leq i \leq n'$, $2 \leq j, \ell, \leq m$, we let $x_{i,j,\ell} = 1$ iff exactly i unestablished voters rank candidate c_j as their ℓ^{th} favorite candidate, i.e., giving them α_ℓ points each.

The constraints are as follows. Each candidate must be given each ranking some $0 \leq i \leq n'$ times. This corresponds to the constraint $\sum_i x_{i,j,\ell} = 1$ for each j, ℓ. Each candidate is ranked exactly n' times, once by each of the manipulators, giving us $\sum_{i,\ell} i x_{i,j,\ell} = n'$ for each j. Each ranking position must also be given exactly n' times, as each manipulator submits a candidate for each ranking position, and thus $\sum_{i,j} i x_{i,j,\ell} = n'$ for each ℓ. Finally, for each j, we ensure that the final score of c_j, $s(c_j) + \sum_{i,\ell} i\alpha_\ell x_{i,j,\ell}$, is at most that of c_1, which is $s(c_1) + n'\alpha_1$. ∎

In this reduction, we are taking advantage of symmetry in that the identities of the voters giving the preferences do not matter; only the cardinalities of the voters giving each type of preference, do. We give an example for an instance of manipulation in an unweighted Borda election as follows.

Consider a Borda election of seven candidates, $c_1, \ldots, c_7$, and three initial voters of preferences $v_1 = (c_7 \succ c_6 \succ c_5 \succ c_4 \succ c_3 \succ c_2 \succ c_1)$, $v_2 = (c_7 \succ c_6 \succ c_4 \succ c_2 \succ c_5 \succ c_3 \succ c_1)$, and $v_3 = (c_5 \succ c_3 \succ c_2 \succ c_1 \succ c_6 \succ c_4 \succ c_7)$. This gives us the initial scores $s(c_1) = 3$, $s(c_2) = s(c_3) = s(c_4) = 8$, and $s(c_5) = s(c_6) = s(c_7) = 12$. Let there be two manipulative voters. c_1 will finish with a score of 15, allowing c_2, c_3, c_4 to each gain a maximum score of 7 and c_5, c_6, c_7 each 3.

There are a total of 108 variables, $x_{i,j,\ell}$ for $0 \leq i \leq 2$, $2 \leq j, \ell \leq 7$, corresponding to whether exactly i of the two manipulative voters rank c_j in the ℓ^{th} position of their preference profile. We examine a case of each of the constraints. Clearly, exactly some number of manipulative voters, between none and two, must rank, for example, c_2 in 2^{nd} place. This gives us the constraint $x_{0,2,2} + x_{1,2,2} + x_{2,2,2} = 1$. c_2 is ranked by exactly two voters, in any position, and thus $0(x_{0,2,2} + \cdots + x_{0,2,7}) + \cdots + 2(x_{2,2,2} + \cdots + x_{2,2,7}) = 2$. There are similar constraints to enforce that each position is given exactly twice, once for each voter.

To ensure the victory of c_1, we compute the final scores as a linear summation based on the positional votes. For example, the final score of c_2 is determined by its initial score and the variables corresponding to it, namely, $x_{i,2,\ell}$ for $0 \leq i \leq 2$ and $2 \leq \ell \leq 7$. Each voter who ranks c_2 in its ℓ^{th} position gives it $(7 - \ell)$ points, and thus, the contribution of $x_{i,2,\ell}$ on the score of c_2 is $i_1(7 - \ell)x_{i,2,\ell}$.

It is also possible to combine both symmetry breaking techniques for the case of manipulating unweighted k-approval elections. However, this problem can be solved in polynomial-time using a greedy algorithm [ZPR09].

5.2 Bribery as a 0-1 Program

0-1 Integer Linear Programming can also be used to model bribery, including cases involving prices and weights. In our first theorem, we assume that no symmetry is involved.

Theorem 5.2.1 *Bribery and \$bribery of the weighted scoring protocols $(\alpha_1, \ldots, \alpha_m)$ of m candidates and n voters can be formulated as a* 0-1 *Integer Linear Programming instance consisting of $\Theta(m^2 n)$ variables and $\Theta(mn)$ constraints.*

Proof: Consider an election of m candidates, $C = \{c_1, \ldots, c_m\}$ and n voters $V = \{v_1, \ldots, v_n\}$. We wish to ensure the victory of c_1 by changing the preferences of voters of total cost at most q. Let $w(v)$ be the weight of voter v, $\pi(v)$ be the cost of bribing voter v, and q be

the total budget of the briber. For each candidate $c \in C$, let $s(c)$ denote its initial score. Further, for $v \in V$ and $c \in C$, let $1 \leq r(v,c) \leq m$ denote the ranking position that v initially gives c.

For $1 \leq i \leq n$, let the variable $x_i = 1$ iff we are to bribe voter v_i, and for $1 \leq i \leq n$ and $2 \leq j, \ell \leq m$, let $y_{i,j,\ell} = 1$ iff we are to bribe voter v_i to rank c_j in the ℓ^{th} position. Without loss of generality, we may assume that all bribed voters are bribed to rank c_1 as their first choice.

If v_i is to be bribed, then we must give each candidate exactly one ranking position and must fill each ranking position exactly once. This corresponds to the equalities $\sum_j y_{i,j,\ell} = x_i$ for all i, ℓ, and $\sum_\ell y_{i,j,\ell} = x_i$ for all i, j.

Because each bribed voter will rank c_1 as their first preference, the final score of c_1 is given by $s(c_1) + \sum_i \left[\alpha_1 - \alpha_{r(v_i,c_1)}\right] w(v_i)x_i$. For $j \neq 1$, the final score of c_j is given by $s(c_j) - \sum_i \alpha_{r(v_i,c_j)} w(v_i)x_i + \sum_{i,\ell} \alpha_\ell w(v_i) y_{i,j,\ell}$.

The budget of the briber is enforced by the inequality $\sum_i \pi(v_i)x_i \leq q$. ■

In our next theorem, we show how symmetry may be broken if the scoring protocol vector contains similar weights, as in the case of k-approval elections. This affects how we represent the new preferences to be given to the bribed voters.

Theorem 5.2.2 *Bribery and $bribery of the weighted k-approval elections of m candidates and n voters can be formulated as a* 0-1 *Integer Linear Programming instance consisting of $\Theta(mn)$ variables and $\Theta(m+n)$ constraints.*

Proof: Consider an election of m candidates, $C = \{c_1, \ldots, c_m\}$ and n voters $V = \{v_1, \ldots, v_n\}$. We wish to ensure the victory of c_1 by changing the preferences of voters of total cost at most q. Let $w(v)$ be the weight of voter v, and $\pi(v)$ be the cost of bribing voter v. For each candidate $c \in C$, let $s(c)$ denote its initial score. Additionally, for each $v \in V$ and $c \in C$, we predefine $\mathsf{app}(v,c)$ to be 1 if v initially approves c, and 0 otherwise.

For $1 \leq i \leq n$, let the variable $x_i = 1$ iff we are to bribe voter v_i, and for $1 \leq i \leq n$ and $2 \leq j \leq m$, let the variable $y_{i,j} = 1$ iff v_i is bribed to approve candidate c_j. Without loss of generality, we may assume that all bribed voters are bribed to approve c_1. By the budget constraint, $\sum_i \pi(v_i)x_i \leq q$, and $\sum_j y_{i,j} = (k-1)x_i$, as $k-1$ additional approvals (besides c_1) must be given to the remaining candidates by the voters to be bribed.

The final score of c_1 is $s(c_1) + \sum_i w(v_i)\,[1 - \mathtt{app}(v_i, c_1)]\,x_i$, while the final score of c_j for $2 \leq j \leq m$ is $s(c_j) - \sum_i w(v_i)\mathtt{app}(v_i, c_j)x_i + \sum_i w(v_i)y_{i,j}$. ■

As in the case of manipulation, symmetry can also be broken when the voters are unweighted. This affects how we represent not only the preferences to be given to the bribed voters, but the initial preferences of the voters we are bribing.

We break symmetry in the initial voters by grouping voters with similar preferences, and counting the number of preferences affected for each case. This is known as succinct preference profile representation, which we described in Section 2.7 (see also [FHH09]).

A bribery can then be represented by counting the number of voters of each distinct preference ordering we are to bribe. This representation is especially useful for problems in which there are many similar voters.

As an example, consider an unweighted Borda election of three candidates a, b, and c, in which 20 voters have a preference of $a \succ b \succ c$, 13 of $b \succ c \succ a$, and 9 of $c \succ a \succ b$. The briber wishes to ensure the victory of c by changing the preferences of at most 10 voters. Using the encoding in the previous theorem, the SAT solver would potentially consider all combinations of at most 10 voters among the set of 42 voters, to bribe. However, it suffices to know how many voters who have each of the three initial preferences are to be bribed, and the preferences to be given to the voters during the bribery. Because this election is unweighted, the same symmetry breaking principles used in the manipulation problem can be applied to the preferences to be given to the bribed voters. We give the general encoding as follows.

Theorem 5.2.3 *Bribery and $bribery of the unweighted scoring protocols $(\alpha_1, \ldots, \alpha_m)$ of m candidates and n voters can also be formulated as a* 0-1 *Integer Linear Programming instance consisting of $\Theta(m^2 n)$ variables and $\Theta(mn)$ constraints.*

Proof: Consider an unweighted election of candidates $C = \{c_1, \ldots, c_m\}$ and voters $V = \{v_1, \ldots, v_n\}$ under the scoring protocol $(\alpha_1, \ldots, \alpha_m)$. We wish to ensure the victory of c_1 by changing the preferences of voters of total cost at most q. For each candidate $c \in C$, let $s(c)$ be the initial score of c. Suppose there are s distinct preferences among the voters V, $P_1, \ldots, P_s$. For each preference P, let $N(P) \geq 1$ denote the number of voters initially having preference profile P. For $0 \leq j \leq N(P)$, let $\Pi(P, j)$ denote the cost of bribing the j cheapest voters of preference P. For $1 \leq i \leq s$ and $0 \leq j \leq N(P_i)$, let variable $x_{i,j} = 1$ iff we bribe exactly j voters with initial preference P_i. For $0 \leq i \leq q$ and $2 \leq j, \ell \leq m$, let $y_{i,j,\ell} = 1$ iff exactly i of the bribed voters are bribed to rank c_j as their ℓ^{th} preference.

Clearly, $\sum_j x_{i,j} = 1$ for all $1 \leq i \leq s$. In total, $\sum_{i,j} j x_{i,j}$ voters are bribed, at a cost of $\sum_{i,j} \Pi(P_i, j) x_{i,j}$, and this quantity must be at most q.

The constraints for the preferences to be given to the bribed voters are similar to the reduction for manipulation, and as follows:

$$\forall j, \ell \sum_i y_{i,j,\ell} = 1$$

$$\forall j \sum_{i,\ell} i y_{i,j,\ell} = \sum_{i,j} j x_{i,j}$$

$$\forall \ell \sum_{i,j} i y_{i,j,\ell} = \sum_{i,j} j x_{i,j}$$

The final score of c_1 is given by $s(c_1) + \sum_{i,j} j \left[\alpha_1 - \alpha_{r(P_i,c_1)}\right] x_{i,j}$, while the final score of c_j for $j \neq 1$ is given by $s(c_j) - \sum_{i,j} j \alpha_{r(P_i,c_j)} x_{i,j} + \sum_{i,j,\ell} i \alpha_\ell y_{i,j,\ell}$.

As in the case of unweighted bribery, we can further break symmetry in scoring protocols with some similar weights.

Consider an unweighted k-approval election of candidates $C = \{c_1, \ldots, c_m\}$ and voters $V = \{v_1, \ldots, v_n\}$. We wish to ensure the victory of c_1 by changing the preferences of voters of total cost at most q. For each candidate $c \in C$, let $s(c)$ be the initial score of c.

Suppose there are s distinct preferences among the voters V, $P_1, \ldots, P_s$. We consider preference similar if they approve the same set of k candidates. For each preference P, let $N(P) \geq 1$ denote the number of voters initially having preference profile P. For $0 \leq j \leq N(P)$, let $\Pi(P, j)$ denote the cost of bribing the j cheapest voters of preference P.

For $1 \leq i \leq s$ and $0 \leq j \leq N(P_i)$, let variable $x_{i,j} = 1$ iff we bribe exactly j voters with initial preference P_i. For $2 \leq i \leq m$ and $0 \leq j \leq q$, we let $y_{i,j} = 1$ iff the bribed voters give exactly j approvals to candidate c_i. Recall that without loss of generality, all bribed voters will approve c_1. The inequality $\sum_{i,j} \Pi(P_i, j) x_{i,j} \leq q$ tells us that we bribe voters of total price at most q. For $2 \leq i \leq m$, candidate c_i receives $\sum_j j y_{i,j}$ approvals. This quantity must be at most the number of voters bribed, or $\sum_{i,j} j x_{i,j}$. The total number of approvals given, given by the quantity $\sum_{i,j} j y_{i,j}$ must equal $k - 1$ times the number of voters bribed.

The final score of candidate c_1 can be given by $s(c_1) - \sum_{i,j} [1 - \mathtt{app}(P_i, c_1)]\, jx_{i,j}$ while that of c_ℓ for $2 \leq \ell \leq m$ by $s(c_\ell) - \sum_{i,j} \mathtt{app}(P_i, c_\ell) jx_{i,j} + \sum_j jy_{i,j}$. ■

In this reduction, briberies affecting the same number of voters of each initial preference are represented by the same instantiation of the variables. Both of these symmetry breaking principles can be combined in the case of bribery of k-approval elections.

5.3 Controlling the Candidate Set as a 0-1 Program

We formulate a reduction from control by adding, deleting, and partitioning candidates to 0-1 Integer Linear Programming. This problem is known to be NP-hard even for very simple election systems such as plurality by a reduction from Hitting Set [BTT92].

Theorem 5.3.1 *Control of weighted k-approval elections of m candidates and n voters by deleting candidates can be formulated as a* 0-1 *Integer Linear Programming instance consisting of $\Theta(mn)$ variables and constraints.*

Proof: Consider a weighted k-approval election of candidates $C = \{c_1, \ldots, c_m\}$ and voters $V = \{v_1, \ldots, v_n\}$, and let c_1 be the distinguished candidate. We wish to ensure the victory of c_1 by deleting at most q candidates.

For each voter $v \in V$, and pair of candidates $c, c' \in C$, we precompute $\mathtt{pref}(v, c, c')$ to be 1 if v prefers c to c' in the initial preference ordering and 0 otherwise. We also precompute $1 \leq \mathtt{pos}(v, c) \leq m$ to be the initial position of c in the preference profile of v. We will use these precomputed variables in the reduction.

For $1 \leq j \leq m$, let the variable $x_j = 1$ iff the candidate c_j is to be deleted. Thus, $\sum_j x_j \leq q$. Also, x_1 clearly must be 0 in a solution. For $1 \leq i \leq n$ and $1 \leq j \leq m$, let the variable $y_{i,j} = 1$ iff v_i will approve c_j following the deletion.

We see that v_i will approve c_j following the deletion iff c_j is not deleted and at least $\mathtt{pos}(v_i, c_j) - k$ candidates ahead of c_j in the preference profile of v_i are deleted; logically, $y_{i,j} \Leftrightarrow \overline{x_j} \wedge \sum_{j' \neq j} \mathtt{pref}(v_i, c_{j'}, c_j) x_{j'} \geq \mathtt{pos}(v_i, c_j) - k$.

The above constraint tells us that both a literal, in this case $\overline{x_j}$, and a particular linear constraint will be satisfied iff a variable, $y_{i,j}$ in this case, is satisfied. Although this is clearly not a 0-1 ILP constraint, using mathematical logic, it is possible to turn this constraint into an equivalent conjunction of 0-1 ILP constraints.

Consider the constraint $x \Leftrightarrow y \wedge (A \leq B)$, where x and y are binary literals and $A \leq B$ is 0-1 ILP constraint. First, it is possible to convert this constraint into a conjunction of the following.

$\overline{x} \vee y$

$\overline{x} \vee (A \leq B)$

$x \vee \overline{y} \vee (B \leq A - 1)$

The first constraint can be written as $y - x \geq 0$. In the second constraint, the inequality $A \leq B$ need only hold if $x = 1$. Consider the inequality $A \leq B + (1 - x)(A_{\max} - B_{\min})$, where $A_{\max}$ is a constant upper bound of A and $B_{\min}$ is a constant lower bound of B. We will discuss what to use for these bounds later. If $x = 0$, this inequality becomes $A \leq B + A_{\max} - B_{\min}$, which is trivially true. If $x = 1$, this inequality becomes $A \leq B$. In the third constraint, the inequality $B \leq A - 1$ must hold if $x = 0$ and $y = 1$, or in other words, if $y - x = 1$. Using the same reasoning from above, this constraint can be represented by the inequality $B \leq A - 1 + (1 + x - y)(B_{\max} - A_{\min} + 1)$, where $A_{\min}$ is a constant lower bound of A and $B_{\max}$ is a constant upper bound of B.

The constraint $x \Leftrightarrow y \wedge (A \leq B)$ can thus be written as a conjunction of three 0-1 ILP constraints. In the above construction, $A_{\min}$, $A_{\max}$, $B_{\min}$, and $B_{\max}$ need to be constant, as they are scaled by the variables x and y. In general, these bounds should be made as tight as possible, to improve pruning by the SAT solver as much as possible.

Recall that above, we wish to convert the constraint $y_{i,j} \Leftrightarrow \overline{x_j} \wedge \sum_{j' \neq j} \texttt{pref}(v_i, c_{j'}, c_j) x_{j'} \geq \texttt{pos}(v_i, c_j) - k$ into a conjunction of linear constraints. In this case, the right hand side of the inequality, $\texttt{pos}(v_i, c_j) - k$, is a constant, and can be used as a lower and upper bound of itself. The left hand side, $\sum_{j' \neq j} \texttt{pref}(v_i, c_{j'}, c_j) x_{j'}$, can be seen to be between 0 and q inclusive, as at most q voters are to be deleted, and thus q of the variables $x_{j'}$ set to 1.

This constraint can be converted into the following linear inequalities.

$x_j + y_{i,j} \leq 1$

$\sum_{j' \neq j} \texttt{pref}(v_i, c_{j'}, c_j) x_{j'} \geq (\texttt{pos}(v_i, c_j) - k) y_{i,j}$

$\sum_{j' \neq j} \texttt{pref}(v_i, c_{j'}, c_j) x_{j'} \leq \texttt{pos}(v_i, c_j) - k - 1 + (x_j + y_{i,j})(q - \texttt{pos}(v_i, c_j) + k + 1)$

This technique allows us to represent linear inequalities that are conditional on one or more variables, and/or vice versa. We will be using similar conversions throughout this chapter.

The final score of each of the candidates can then be computed as a linear function of the variables $y_{i,j}$. The final score of candidate c_j is given by the sum $\sum_{1 \leq i \leq n} w(v_i) y_{i,j}$. To ensure the victory of c_1, for each $2 \leq j \leq m$, $\sum_{1 \leq i \leq n} w(v_i) y_{i,1} \geq \sum_{1 \leq i \leq n} w(v_i) y_{i,j}$. ■

Theorem 5.3.2 *We may break symmetry by consolidating voters of similar preferences, so that there are at most* $\frac{m!}{(m-k-q)!k!}$ *voters, in problems of control by deleting* q *candidates in* k*-approval elections of* m *candidates and* n *voters.*

Proof: We note that in a k-approval election, for a candidate to be approved by a voter following a deletion of q candidates, it is necessary for the candidate to be among the top $k+q$ candidates of the voter's preference. Thus, only the top $k+q$ candidates of each voter's preference ordering are of importance. Furthermore, unless they are deleted, the top k candidates of a voter's preference are approved. The order of the top k candidates thus is of no importance. There are thus $\frac{m!}{(m-k-q)!k!}$ distinguishable preferences in this problem. We may consolidate indistinguishable voter preferences by combining their weights prior to the reduction to 0-1 Integer Linear Programming.

Although this simplification, unlike in the cases of manipulation and bribery, does not eliminate similar deletions of candidates, it reduces the complexity of determining the outcome of each deletion. This simplification also places a cap on the problem size when the size of the candidate set is bounded, regardless of the size of the voter set. The problem of control by deleting candidates does not appear to have any trivial symmetries. ■

Theorem 5.3.3 *Control of weighted* k*-approval elections of* m *candidates and* n *voters by adding candidates can be formulated as a* 0-1 *Integer Linear Programming instance consisting of* $\Theta(mn)$ *variables and constraints.*

Proof: Consider a weighted k-approval election of candidates $C \cup C' = \{c_1, \ldots, c_m, c'_1, \ldots, c'_{m'}\}$ and voters $V = \{v_1, \ldots, v_n\}$, and let c_1 be the distinguished candidate. We wish to ensure the victory of c_1 by adding at most q candidates to the initial candidate set of $C = \{c_1, \ldots, c_m\}$.

For each voter $v \in V$, and pair of candidates $c, c' \in C \cup C'$, we precompute $\texttt{pref}(v, c, c')$ be 1 if v prefers c to c' and 0 otherwise. For each $v \in V$ and $c \in C$, let $1 \leq \texttt{pos}(v, c) \leq m$ be the initial position of c in the preference ordering of v. We will use these precomputed variables in the reduction.

For $1 \leq j \leq m'$, let the variable $x_j = 1$ iff we are to add the candidate c'_j. Clearly, $\sum_j x_j \leq q$. For $1 \leq i \leq n$ and $1 \leq j \leq m$, let the variable $y_{i,j} = 1$ iff v_i approves c_j following

the addition, and for $1 \leq i \leq n$ and $1 \leq j' \leq m'$, $y'_{i,j} = 1$ iff v_i approves c'_j following the addition.

We see that v_i approves c_j iff at most $k - \texttt{pos}(v_i, c_j)$ candidates ahead of c_j in the preferences of v_i are added. In other words, $y_{i,j} \Leftrightarrow \sum_{j'} \texttt{pref}(v_i, c'_{j'}, c_j) x_{j'} \leq k - \texttt{pos}(v_i, c_j)$. We can rewrite this constraint as a conjunction of the following linear constraints (see Theorem 5.3.1 for explanation of equivalence).

$$\sum_{j'} \texttt{pref}(v_i, c'_{j'}, c_j) x_{j'} \leq k - \texttt{pos}(v_i, c_j) + (1 - y_{i,j})(q - k + \texttt{pos}(v_i, c_j))$$

$$\sum_{j'} \texttt{pref}(v_i, c'_{j'}, c_j) x_{j'} \geq (k - \texttt{pos}(v_i, c_j) + 1)(1 - y_{i,j})$$

Similarly, we see that v_i approves c'_j iff c'_j is added and at most a total of $k-1$ candidates of the initial candidate set and of added candidates ahead of c'_j in the preferences of v_i are in the final election. This is equivalent to the following linear constraints.

$$y'_{i,j} - x_j \leq 0$$

$$\sum_{j'} \texttt{pref}(v_i, c'_{j'}, c'_j) x_{j'} + \sum_{j'} \texttt{pref}(v_i, c_{j'}, c'_j) \leq k - 1 + (1 - y'_{i,j})(m + q - k)$$

$$\sum_{j'} \texttt{pref}(v_i, c'_{j'}, c'_j) x_{j'} + \sum_{j'} \texttt{pref}(v_i, c_{j'}, c'_j) \geq k(x_j - y_{i,j})$$

The final scores of the candidates can then be computed as a linear function of the variables $y_{i,j}$ and $y'_{i,j}$. c_1 must beat each of the candidates c_j for $2 \leq j \leq m$, as well as any candidates c'_j for $1 \leq j \leq m'$ which is added. Thus, for each $2 \leq j \leq m$, $\sum_{1 \leq i \leq n} w(v_i) y_{i,1} \geq \sum_{1 \leq i \leq n} w(v_i) y_{i,j}$, and for each $1 \leq j \leq m'$, $\sum_{1 \leq i \leq n} w(v_i) y_{i,1} \geq \sum_{1 \leq i \leq n} w(v_i) y'_{i,j}$. ■

Theorem 5.3.4 *We may break symmetry by consolidating voters of similar preferences, so that there are at most* $\frac{m!(m'+k)!}{(k!)(m-k)!}$ *voters, in problems of control by adding* q *candidates in* k*-approval elections of* m *candidates and* n *voters, in which there are* m' *additional candidates.*

Proof: As we are adding candidates, only the first k candidates of each voter's preference ordering may be approved by the voter following the addition of candidates. It suffices to consider the order of these k candidates and the m' additional candidates among the voter's preference. There are $\frac{m!(m'+k)!}{(k!)(m-k)!}$ distinguishable preferences in this problem. We may consolidate indistinguishable voter preferences by combining their weights. As in the case of deleting candidates, this simplification does not remove symmetry in the problem itself, but instead, simplifies the counting of the final score within the encoding. ■

Theorem 5.3.5 *Control of weighted k-approval elections of m candidates and n voters by partitioning candidates can be formulated as a* 0-1 *Integer Linear Programming instance consisting of* $\Theta(m^2 n)$ *variables and constraints.*

Proof: Consider a weighted k-approval election of candidates $C = \{c_1, \ldots, c_m\}$ and voters $V = \{v_1, \ldots, v_n\}$, and let c_1 be the distinguished candidate. We wish to ensure the victory of c_1 by partitioning the set of candidates.

For each voter $v \in V$, and pair of candidates $c, c' \in C$, we precompute $\texttt{pref}(v, c, c')$ be 1 if v prefers c to c' in the initial preference ordering and 0 otherwise. We will use these precomputed variables in the reduction.

For $1 \leq j \leq m$, let the variable x_j denote the set in which candidate c_j is to be placed (call the subsets 0 and 1 here). Without loss of generality, we may add the constraint that c_1 is placed into set 0, giving $x_1 = 0$. For $s \in \{0, 1\}$, $1 \leq i \leq n$, $1 \leq j \leq m$, let the variable $\texttt{app}^s_{i,j} = 1$ if voter v_i approves c_j in set s. This occurs iff c_j is placed in subset s and at most $k-1$ candidates ahead of c_j in the preference profile of v_i are placed in the same subset.

This is given by the constraint $\texttt{app}^0_{i,j} \Leftrightarrow \overline{x_j} \wedge \left[\sum_{j'} \texttt{pref}(v_i, c_{j'}, c_j)(1 - x_{j'}) \leq k - 1\right]$ for $s = 0$ and $\texttt{app}^1_{i,j} \Leftrightarrow x_j \wedge \left[\sum_{j'} \texttt{pref}(v_i, c_{j'}, c_j) x_{j'} \leq k - 1\right]$ for $s = 1$.

Using obvious lower and upper bounds of these summations, we can rewrite these constraints linearly as follows (see Theorem 5.3.1 for explanation of equivalence).

$\texttt{app}^0_{i,j} + x_j \leq 1$

$\sum_{j'} \texttt{pref}(v_i, c_{j'}, c_j)(1 - x_{j'}) \leq k - 1 + (1 - \texttt{app}^0_{i,j})(m - k)$

$\sum_{j'} \texttt{pref}(v_i, c_{j'}, c_j)(1 - x_{j'}) + (\texttt{app}^0_{i,j} + x_j)k \geq k$

$\texttt{app}^1_{i,j} - x_j \leq 0$

$\sum_{j'} \texttt{pref}(v_i, c_{j'}, c_j) x_{j'} \leq k - 1 + (1 - \texttt{app}^1_{i,j})(m - k)$

$\sum_{j'} \texttt{pref}(v_i, c_{j'}, c_j) x_{j'} + (\texttt{app}^1_{i,j} + 1 - x_j)k \geq k$

The score of each candidate c_j in set s can be computed as the summation $\sum_i \texttt{app}^s_{i,j}$.

For $s \in \{0, 1\}$ and $1 \leq j, j' \leq m$, let the variable $\texttt{beats}^s_{j,j'} = 1$ iff c_j beats or ties $c_{j'}$ in a k-approval election of subset s. This occurs iff $\sum_i \texttt{app}^s_{i,j} \geq \sum_i \texttt{app}^s_{i,j'}$. This is equivalent to the following linear constraints.

$$\sum_i \texttt{app}^s_{i,j} + (1 - \texttt{beats}^s_{j,j'}) \sum_i w(v_i) \geq \sum_i \texttt{app}^s_{i,j'}$$

$$\sum_i \texttt{app}^s_{i,j} \leq \sum_i \texttt{app}^s_{i,j'} - 1 + \texttt{beats}^s_{j,j'} \left[\sum_i w(v_i) + 1 \right]$$

For $s \in \{0, 1\}$ and $1 \leq j \leq m$, let the variable $w^s_j = 1$ iff c_j is a winner of the k-Approval election of subset s. A candidate is a winner iff it beats or ties all other candidates. Thus, $w^s_j \Leftrightarrow \texttt{beats}^s_{j,1} \wedge \cdots \wedge \texttt{beats}^s_{j,m}$. This can be represented by the linear constraints $\texttt{beats}^s_{j,j'} - w^s_j$ for each $1 \leq j, j' \leq m$ and $w^s_j - \sum_j beats^s_{j,j'} \geq 1 - m$ for each $1 \leq j \leq m$.

Let the variable $w_j = 1$ iff c_j is a winner of either subelection, or $w_j \Leftrightarrow w^0_j \vee w^1_j$. This can be represented by the linear constraints $w^0_j + w^1_j - w_j \geq 0$, $w_j - w^0_j \geq 0$ and $w_j - w^1_j \geq 0$ for each $1 \leq j \leq m$.

Clearly, c_1 must win at least one of the subelections for the manipulation to succeed, and thus we have the constraint $w_1 = 1$.

For $1 \leq i \leq n$ and $1 \leq j, j' \leq m$, let the variable $\texttt{pref}^2_{i,j,j'} = 1$ iff both c_j and $c_{j'}$ are in the second round of the election and v_i prefers c_j to $c_{j'}$, and the variable $\texttt{app}^2_{i,j} = 1$ iff v_i approves c_j in the final round. This occurs iff c_j is among the final round, represented by variable w_j, and that at most $k - 1$ candidates of the final round are preferred to c_j, or $\sum_{j'} \texttt{pref}^2_{i,j',j} \leq k - 1$. This can be rewritten as the following linear constraints.

$$\texttt{app}^2_{i,j} - w_j \leq 0$$

$$\sum_{j'} \texttt{pref}^2_{i,j',j} \leq k - 1 + (m - k)(1 - \texttt{app}^2_{i,j})$$

$$\sum_{j'} \texttt{pref}^2_{i,j',j} \geq k(w_j - \texttt{app}^2_{i,j})$$

This allows us to compute the final score of each candidate in the final round. The final score of c_j, for $2 \leq j \leq m$ if it is in the final round, is $\sum_i w(v_i)\texttt{app}^2_{i,j}$, and must be at most that of c_1, which is $\sum_i w(v_i)\texttt{app}^2_{i,1}$.

For this problem, in both the weighted and unweighted cases, we may consolidate voters with identical preferences by combining their weights (1 for unweighted voters). While this does not break symmetry for the partitioning of the candidates (there are no trivial symmetries in this case), such a simplification reduces the complexity of computing the final score.

This reduction also demonstrates how one may encode problems in multi-round election systems into 0-1 ILP instances. ■

We give an example as follows.

Consider a plurality election of candidates $C = \{c_1, c_2, c_3, c_4\}$ and voters $V = \{v_1, v_2, v_3, v_4, v_5\}$ of weights 18, 9, 5, 3, and 2 respectively, with the following preferences: $v_1 = (c_4 \succ c_1 \succ c_2 \succ c_3)$, $v_2 = (c_1 \succ c_2 \succ c_3 \succ c_4)$, $v_3 = (c_2 \succ c_3 \succ c_4 \succ c_1)$, $v_4 = (c_3 \succ c_4 \succ c_1 \succ c_2)$, and $v_5 = (c_3 \succ c_4 \succ c_1 \succ c_2)$.

First, we may combine v_4 and v_5, having identical preferences, giving us an election of four candidates $C = \{c_1, c_2, c_3, c_4\}$ and four voters $V = \{v_1, v_2, v_3, v_4\}$ of weights 18, 9, 5, and 5 respectively, with preferences $v_1 = (c_4 \succ c_1 \succ c_2 \succ c_3)$, $v_2 = (c_1 \succ c_2 \succ c_3 \succ c_4)$, $v_3 = (c_2 \succ c_3 \succ c_4 \succ c_1)$, and $v_4 = (c_3 \succ c_4 \succ c_1 \succ c_2)$.

With no manipulation, c_4 is the unique winner with a score of 18. However, it is possible to make c_1 a winner by partitioning the candidates as follows: $C = \{c_1, c_2\} \cup \{c_3, c_4\}$. c_1 wins in an election of candidates $\{c_1, c_2\}$, while c_3 wins an election of candidates $\{c_3, c_4\}$. c_1 then wins the final round against c_3.

In our reduction, we use variables x_1,x_2,x_3, and x_4 as binary variables to represent the sets we partition the candidates c_1, c_2, c_3, and c_4, respectively, into, calling the sets 0 and 1. Without loss of generality, c_1 will be placed in set 0, and $x_1 = 0$. Consider the boolean variable $\texttt{app}^0_{1,1}$ in our reduction. This variable tells us whether or not v_1 approved c_1 in subset 0 of the partition. This is true iff c_1 is placed in subset 0 (corresponding to $x_1 = 0$), and at most 0 of the candidates that v_1 prefers to c_1 (c_4 in this case) is placed in this subset, as we are looking at plurality, or 1-approval. Thus, $\texttt{app}^0_{1,1} = 1$ iff $x_1 = 0$ or $x_4 \leq 0$. In our reduction, this is handled by the following constraints.

$\texttt{app}^0_{1,1} + x_1 \leq 1$

$1 - x_4 \leq 3(1 - \texttt{app}^0_{1,1})$

$1 - x_4 + (\texttt{app}^0_{1,1} + x_1) \geq 1$

Logically, the first constraint tells us that v_1 cannot approve c_1 in subset 0 if c_1 is not placed in subset 0. The second constraint tells us that if v_1 approves c_1 in subset 0, then no more than zero of the candidates v_1 prefers to c_1 (c_4 only) can be placed in subset 0. The final constraint tells us that if v_1 is placed in subset 0 and no more than zero of those v_1 prefers to c_1 is, then v_1 approves c_1 in this subset. The other variables $\texttt{app}^s_{i,j}$ for $s \in \{0, 1\}$ and $1 \leq i \leq n, 1 \leq j \leq m$ are defined similarly.

Consider the variable $\texttt{beats}^0_{1,2}$. This variable tells us that candidate c_1 beats or ties candidate c_2 in subset 0 of the partition. This occurs iff $\sum_{1 \leq i \leq 4} w(v_i)\texttt{app}^0_{i,1} \geq \sum_{1 \leq i \leq 4} w(v_i)\texttt{app}^0_{i,2}$.

In our reduction, this is represented by two linear constraints.

$$\sum_{1 \leq i \leq 4} w(v_i)\texttt{app}^0_{i,1} + 37(1 - \texttt{beats}^0_{1,2}) \geq \sum_{1 \leq i \leq 4} w(v_i)\texttt{app}^0_{i,2}$$

$$\sum_{1\leq i\leq 4} \mathtt{app}^0_{i,1} \leq \sum_{1\leq i\leq 4} \mathtt{app}^0_{i,2} - 1 + 38\mathtt{beats}^0_{1,2}$$

Note that 37 is the sum of the weights of the voters, which represents an upper bound of each score. Logically, the first inequality tells us that if $\mathtt{beats}^0_{1,2} = 1$, then the score of c_1 in set 0 is at least that of c_2, while the second inequality tells us that if $\mathtt{beats}^0_{1,2} = 0$, this is not the case.

Theorem 5.3.6 *Similar* 0-1 *ILP encodings can be made for the problems of control by deleting, adding, and partitioning candidates in Borda elections of weighted voters.*

Proof: Consider a weighted k-approval election of candidates $C = \{c_1, \ldots, c_m\}$ and voters $V = \{v_1, \ldots, v_n\}$, and let c_1 be the distinguished candidate. We wish to ensure the victory of c_1 by deleting at most q candidates. For each pair of candidates $c, c' \in C$, let $\mathtt{pref}(c, c')$ be the sum of the weights of voters preferring c to c' initially. These parameters can be precomputed, and we will use this setup for all three reductions (adding, deleting, and partitioning candidates). By the definition of the Borda count election, the initial score of c is thus $\sum_{c'\neq c} \mathtt{pref}(c, c')$.

For $1 \leq i \leq m$, let the variable $x_i = 1$ iff we are to delete the candidate c_i. Clearly, $\sum_i x_i \leq q$ and $x_1 = 0$. The final score of c_1 is given by $\sum_{j\neq 1} \mathtt{pref}(c_1, c_j)x_j$, while that of c_i (if not deleted) is given by $\sum_{j\neq i} \mathtt{pref}(c_i, c_j)x_j$. This, if $x_i = 0$, then it must be the case that

$$\sum_{j\neq 1, j\neq i} \left[\mathtt{pref}(c_i, c_j) - \mathtt{pref}(c_1, c_j)\right] x_j \leq 0.$$

This constraint can be written as $\sum_{j\neq 1, j\neq i} \left[\mathtt{pref}(c_i, c_j) - \mathtt{pref}(c_1, c_j)\right] x_j \leq (k-1)_i w(v_i)x_i$.

In the case of control by adding candidates, we are given an additional set of candidates $C' = \{c'_1, \ldots, c'_{m'}\}$, and wish to ensure the victory of v_1 by adding at most q candidates among this set. In this case, the final score of c_i is given by $\sum_{j\neq i} \mathtt{pref}(c_i, c_j) + \sum_j \mathtt{pref}(c_i, c'_j)x_j$, while the final score of c'_i (if added) is given by $\sum_j \mathtt{pref}(c_i, c_j) + \sum_{j\neq i} \mathtt{pref}(c_i, c'_j)x_j$. c_1 must score at least the score of each c_i for $2 \leq i \leq m$, and at least the score of each c'_i if it is added (if $x_i = 1$). The latter constraint can be written as follows.

$$\sum_j \left[\mathtt{pref}(c'_i, c'_j) - \mathtt{pref}(c_1, c'_j)\right] x_j \leq (k-1)\sum_i w(v_i) + \left[\sum_{j\neq 1} \mathtt{pref}(c'_i, c_j) - \sum_j \mathtt{pref}(c'_i, c_j) + (k-1)\sum_i w(v_i)\right] x_i$$

For partitioning candidates, let the variable x_i denote the set in which c_i is to be placed in (call them 0 and 1). Without loss of generality, we may assume that c_1 is to be placed in set 0, and this $x_i = 0$.

If c_i is placed in set 0, then the final score of c_i in set 0 is given by $\sum_{j \neq i} \texttt{pref}(c_i, c_j)(1 - x_i)$. Similarly, the final score of c_i in set 1 if placed there is $\sum_{j \neq i} \texttt{pref}(c_i, c_j)x_i$. For convenience we denote these sums $\texttt{score}_i^0$ and $\texttt{score}_i^1$ respectively (note that these are not additional variables).

As we did in the cases of k-approval, we let the variable $\texttt{beats}_{i,j}^s = 1$ iff c_i and c_j are both to be placed in set s and c_i beats or ties c_j in the corresponding subelection. Logically, we write

$\texttt{beats}_{i,j}^0 \Leftrightarrow \overline{x_i} \wedge \overline{x_j} \wedge \texttt{score}_i^0 \geq \texttt{score}_j^0$.

$\texttt{beats}_{i,j}^1 \Leftrightarrow x_i \wedge x_j \wedge \texttt{score}_i^1 \geq \texttt{score}_j^1$.

This is equivalent to the following linear constraints:

$\texttt{beats}_{i,j}^0 + x_i \leq 1$

$\texttt{beats}_{i,j}^0 + x_j \leq 1$

$\texttt{score}_j^0 \leq \texttt{score}_i^0 + (1 - \texttt{beats}_{i,j}^0)(k-1)$

$\texttt{score}_i^0 \leq \texttt{score}_j^0 - 1 + (x_i + x_j + \texttt{beats}_{i,j}^0)(k-1)\sum_i w(v_i)$

$\texttt{beats}_{i,j}^1 - x_i \leq 0$

$\texttt{beats}_{i,j}^1 - x_j \leq 0$

$\texttt{score}_j^1 \leq \texttt{score}_i^1 + (1 - \texttt{beats}_{i,j}^1)(k-1)$

$\texttt{score}_i^1 \leq \texttt{score}_j^1 - 1 + (2 - x_i - x_j + \texttt{beats}_{i,j}^1)(k-1)\sum_i w(v_i)$

As in the case of partitioning candidates in k-approval elections, we define the variables w_j^s, for $s \in \{0, 1\}$ and $1 \leq j \leq m$, such that $w_j^s = 1$ iff c_j is a winner in the subelection of set s. Similarly, for $1 \leq j \leq m$, let variable $w_j = 1$ iff c_j is a winner of either subelection, and for $1 \leq i \leq n$ and $1 \leq j, j' \leq m$, $\texttt{pref}_{i,j,j'}^2 = 1$ iff both c_j and $c_{j'}$ are a winner of either subelection, making it to the final round, and v_i prefers c_j to $c_{j'}$. The constraints defining these variables are identical to that of Theorem 5.3.5.

For each $1 \leq j \leq m$, the final score of c_j is given by $\sum_{i,j'} w(v_i)\texttt{pref}_{i,j,j'}^2$. Thus, for $2 \leq j \leq m$, it must be the case that $\sum_{i,j'} w(v_i)\texttt{pref}_{i,1,j'}^2 \geq \sum_{i,j'} w(v_i)\texttt{pref}_{i,j,j'}^2$, ensuring the victory of c_1. ∎

5.4 Controlling the Voter Set as a 0-1 Program

The encoding for control by adding or deleting voters into 0-1 ILP is similar to the case of bribery: Variables tell us the set of voters to add or delete.

Theorem 5.4.1 *Control of weighted k-approval elections of m candidates and n voters by adding voters from a set of n' voters can be formulated as a* 0-1 *Integer Linear Programming instance consisting of $\Theta(n')$ variables and $\Theta(m)$ constraints.*

Proof: Consider a weighted k-approval election of candidates $C = \{c_1, \ldots, c_m\}$ and voters $V = \{v_1, \ldots, v_n\}$, and let c_1 be the distinguished candidate. Let $V' = \{v'_1, \ldots, v'_{n'}\}$ be an additional set of voters in which we wish to add at most q voters, ensuring the victory of c_1.

We precompute some variables as follows. For each $v \in V'$ and $c \in C$, let $\texttt{app}(v, c)$ be 1 if v approves c and 0 otherwise. Let $s(c)$ denote the initial score of candidate c.

For each $1 \leq i \leq n'$, let the variable $x_i = 1$ iff we are to add the voter v'_i to the election. Since we wish to add at most q candidates, $\sum_i x_i \leq q$. For $2 \leq j \leq m$, the final score of candidate c_j is given by $s(c_j) + \sum_i w(v'_i) app(v'_i, c_j)$, which must be at most that of c_1, which is $s(c_1) + \sum_i w(v'_i) app(v'_i, c_1)$. ∎

Theorem 5.4.2 *Control of weighted k-approval elections of m candidates and n voters by deleting at most q voters can be formulated as a* 0-1 *Integer Linear Programming instance consisting of $\Theta(n)$ variables and $\Theta(m)$ constraints.*

Proof: Consider a weighted k-approval election of candidates $C = \{c_1, \ldots, c_m\}$ and voters $V = \{v_1, \ldots, v_n\}$, and let c_1 be the distinguished candidate. We wish to ensure the victory of c_1 by deleting at most q candidates. For each $v \in V$ and $c \in C$, let $\texttt{app}(v, c)$ be 1 if v initially approves c and 0 otherwise.

For $1 \leq i \leq n$, let $x_i = 1$ iff the voter v_i is to be deleted. Clearly, $\sum_i x_i \leq q$. For $2 \leq j \leq m$, the final score of c_j is given by the sum $\sum_i w(v_i)\texttt{app}(v_i, c_j)(1 - x_i)$, and must be at most that of c_1, which is $\sum_i w(v_i)\texttt{app}(v_i, c_1)(1 - x_i)$. ∎

In the problem of control by adding or deleting unweighted voters in k-approval elections, one may break symmetry by representing the voters to be affected (in the case of adding voters, the additional set of voters, and in the case of deleting voters, the initial set of

voters) succinctly, giving us at most $\binom{m}{k}$ distinct preferences. The encodings would count the number of voters of each distinct preference to be added or deleted. This is similar to that of Theorem 5.2.3.

Theorem 5.4.3 *Control of weighted k-approval elections of m candidates and n voters by partitioning voters can be formulated as a* 0-1 *Integer Linear Programming instance consisting of $\Theta(m^2 n)$ variables and constraints.*

Proof: Consider a weighted k-approval election of candidates $C = \{c_1, \ldots, c_m\}$ and voters $V = \{v_1, \ldots, v_n\}$, and let c_1 be the distinguished candidate. For each $v \in V$ and $c \in C$, let $\mathtt{app}(v, c)$ be 1 if v initially approves c and 0 otherwise.

We wish to partition the voters V into subsets two subsets $V = V_1 \cup V_2$, such that the c_1 is a winner of an election among the winners of the subelections (C, V_1) and (C, V_2).

For $1 \leq i \leq m$, let x_i represent the subset in which v_i is placed in (call the subsets 0 and 1 here). Without loss of generality, we assume that v_1 is placed in subset 0, and thus $x_1 = 0$. The score of c_j in the subelection of subset 0 is given by the sum $\sum_i \mathtt{app}(v_i, c_j)(1 - x_i)$, and that in subset 1 is given by $\sum_i \mathtt{app}(v_i, c_j) x_i$.

For $s \in \{0, 1\}$, $1 \leq j \neq j' \leq m$, let the variable $\mathtt{beats}^s_{j,j'} = 1$ iff c_j beats or ties c'_j in the subelection of subset s. For subset 0, we can logically write

$$\mathtt{beats}^0_{j,j'} = 1 \Leftrightarrow \sum_i \left[\mathtt{app}(\mathtt{v_i}, \mathtt{c_j}) - \mathtt{app}(\mathtt{v_i}, \mathtt{c_{j'}})\right] \mathtt{w}(\mathtt{v_i})(\mathtt{1} - \mathtt{x_i}) \geq 0.$$

This is equivalent to the following two linear constraints:

$$\sum_i \left[\mathtt{app}(\mathtt{v_i}, \mathtt{c_j}) - \mathtt{app}(\mathtt{v_i}, \mathtt{c_{j'}})\right] \mathtt{w}(\mathtt{v_i})(\mathtt{1} - \mathtt{x_i}) \leq (\mathtt{beats}^0_{j,j'} - 1) \sum_i w(v_i)$$

$$\sum_i \left[\mathtt{app}(\mathtt{v_i}, \mathtt{c_j}) - \mathtt{app}(\mathtt{v_i}, \mathtt{c_{j'}})\right] \mathtt{w}(\mathtt{v_i})(\mathtt{1} - \mathtt{x_i}) \geq \sum_i w(v_i) - \left[\sum_i w(v_i) + 1\right] (1 - \mathtt{beats}^0_{j,j'})$$

The remainder of this reduction is identical to that of Theorem 5.3.5: we compute the scores linearly to find the winners of each subelection and compute the final winners among the winners of both subelections. See the definition and constraints for the variables w^s_j, w_j, and $\mathtt{pref}^2_{i,j,j'}$ in Theorem 5.3.5. ∎

Theorem 5.4.4 *Symmetry can be broken when the voters are unweighted using succinct preference profile representation.*

Proof: Consider a weighted k-approval election of candidates $C = \{c_1, \ldots, c_m\}$ and voters $V = \{v_1, \ldots, v_n\}$, and let c_1 be the distinguished candidate. Suppose there are s distinct preference profiles, call them $P_1, \ldots, P_s$, and for $P \in \{P_1, \ldots, P_s\}$ let $N(P) \geq 1$ denote the

number of voters having preference P. For each $P \in \{P_1, \ldots, P_s\}$ and $c \in C$, let $\texttt{app}(P, c)$ be 1 if voters of preference P initially approves c and 0 otherwise.

We wish to partition the voters V into subsets two subsets $V = V_1 \cup V_2$, such that the c_1 is a winner of an election among the winners of the subelections (C, V_1) and (C, V_2).

For each $1 \leq j \leq s$ and $0 \leq i \leq N(P_j)$, let the variable $x_{i,j} = 1$ iff exactly i voters of preference P_j are to be placed in V_1 (and exactly $N(P_j) - i$ voters of preference P_j are to be placed in V_2).

The score of c_j in subelection (C, V_1) is then given by the sum $\sum_{i,j} i\texttt{app}(P_i, c_j)x_{i,j}$, while the score in subelection (C, V_2) is given by $\sum_{i,j} [N(P_j) - i]\texttt{app}(P_i, c_j)x_{i,j}$. The remainder of the construction is identical to that of weighted cases: Recall that we use variables $\texttt{beats}^s_{j,j'}$ to indicate whether c_j will beat $c_{j'}$ in the subelection of set s, and w^s_j to indicate whether c_j will be a winner in set s. Then we can compute the preferences of the final round using variables $\texttt{pref}^2_{i,j,j'}$ and compute the final score of each candidate, ensuring that c_1 is a winner.

■

5.5 Experimental Results

In this section, we test the feasibility of using these reductions and Clasp SAT Solver to solve manipulation problems of interest.

Let n be the number of established voters and m be the number of unestablished manipulators. From the known phase transition and minimal coalition results for manipulation [Wal09, Con10], and phase transition of the problem of Partition [GW96], we know that hard instances of manipulation occur at the phase transition of $m = \Theta(\sqrt{n})$ and, in the case of solving using algorithms for Partition, when the weights of the voters are uniformly distributed in $[1, \Theta(2^m)]$. Recall from Section 2.9 that a phase transition occurs in most NP-hard problems, in which there are just enough constraints to make solutions hard to find, but not enough to easily reach a contradiction, and hard instances often occur in this area. We examine the runtime complexity of invoking the Clasp algorithm for these cases.

In each case, as a benchmark, we count the number of conflicts encountered by Clasp in the SAT instances, and whenever possible, compute a 95% confidence interval, to ensure statistical significance. In SAT solver algorithms, a conflict occurs when the algorithm backtracks. This is similar to the counting of branches by [Wal09] in the evaluation of computing manipulations of veto elections of three candidates, which we have also used in

Chapter 4 for manipulating scoring protocols using algorithms of Partition. This benchmark is of interest because it ignores the depth of the search tree, which is largely a function of the problem size, allowing one to focus on the phase transition of the problem, which is independent of the problem size. There are many other interesting benchmark statistics returned by most modern SAT solvers, including restarts, learnt clauses, forgotten clauses, and decisions (see "The international SAT Competitions" [sat]).

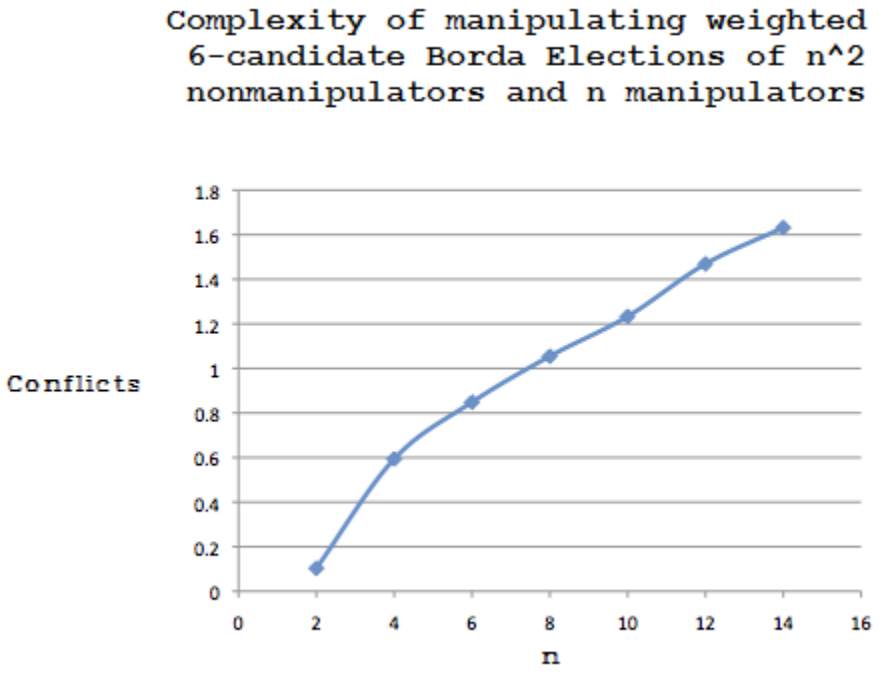
In weighted problems, we choose the weights of the voters independently and uniformly from the range $[0, 2^m]$, where m is the number of manipulators and preference orderings uniformly as a random permutation. The distribution of each random voter's preferences are independent, and a shuffle algorithm ensuring that each preference has a probability of $\frac{1}{m!}$ of being chosen is used.

A shuffle algorithm outputs a permutation of m items such that the probability of each permutation of is exactly $\frac{1}{m!}$. A common shuffle algorithm is that of the Fisher-Yates shuffle. Starting with any ordering, each element is iterated, and swapped with a random position that has not yet been passed (including itself). For example, suppose we wish to shuffle the elements $\{1, 2, 3, 4\}$. Element 1 may be swapped with any of the four elements, each with probability $\frac{1}{4}$. Suppose it is swapped with 3, giving the new permutation $\{3, 2, 1, 4\}$. Element 2 may be swapped with one of the elements 2, 1, or 4, each with probability $\frac{1}{3}$. Supposed it is swapped with 1, giving $\{3, 1, 2, 4\}$. In the 3^{rd} position, 2 may be swapped with either 2 or 4, each with probability $\frac{1}{2}$. Suppose it is swapped with itself, leaving the permutation unchanged. 4 may only be swapped with itself, giving us a shuffle of $\{3, 1, 2, 4\}$. The probability of each permutation being outputted is $\frac{1}{4}\frac{1}{3}\frac{1}{2}\frac{1}{1} = \frac{1}{4!}$, as each permutation can result from only one choice of random variables.

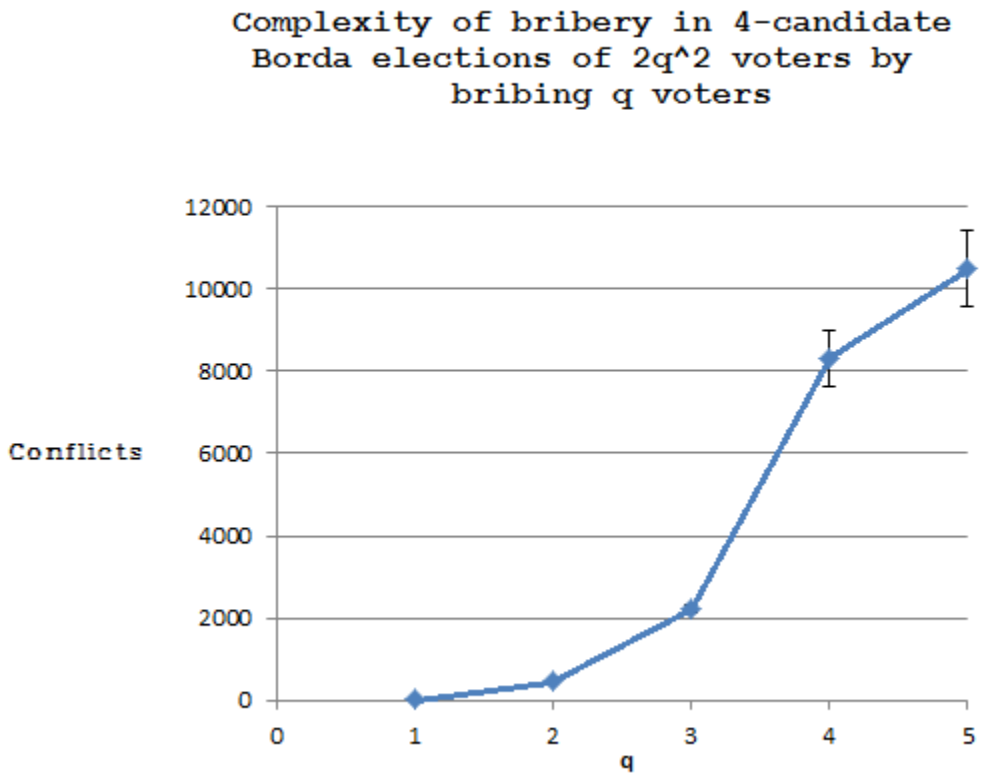
Recall that from Chapter 4 that in [Wal09], the weights of the voters in the random elections tested were chosen from a fixed interval, $[1, 256]$, and that the resulting problem is, theoretically, polynomial-time computable. However, due to the large exponent of this theoretical polynomial-time result and the size of the problems tested, this is unlikely to significantly affect the empirical results given, as it would be impractical to capitalize on the theoretical polynomial-time algorithm.

Our first test is for the case of weighted scoring protocols of a fixed candidate size. In Chapter 4, it is shown that natural extensions of algorithms for Partition were unable to extend the results of [Wal09] of three-candidate scoring protocols, to nontrivial scoring protocols of four or more candidates. We find that Clasp is able to solve manipulation problems of weighted Borda for four, five, and six candidates with a linear number of conflicts, even

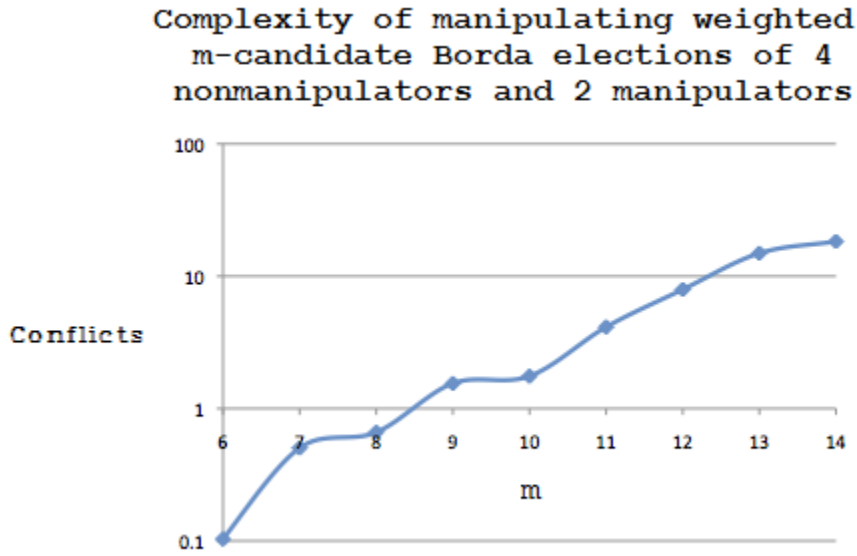
within the phase transition for manipulation as well as with integers chosen in the range of the phase transition for the Partition problem, extending the results of Walsh for these cases. We presume that this result holds for all scoring protocols of a fixed number of candidates. We plot the results for manipulating six-candidate weighted Borda elections in the phase transition of the problem as follows.



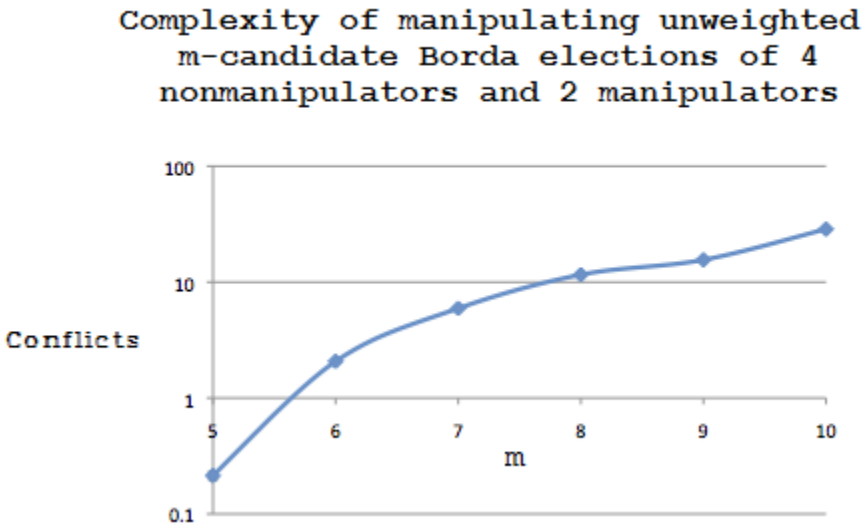
The same easiness within the phase transition is also found for the case of weighted bribery. We show an example in the case of bribery in weighted 4-candidate Borda elections.



We examine the family of scoring protocols Borda for weighted elections. In essence, we want to see if this complexity also scales well when the number of candidates is increased, just as when the number of voters (both manipulative and nonmanipulative) are.



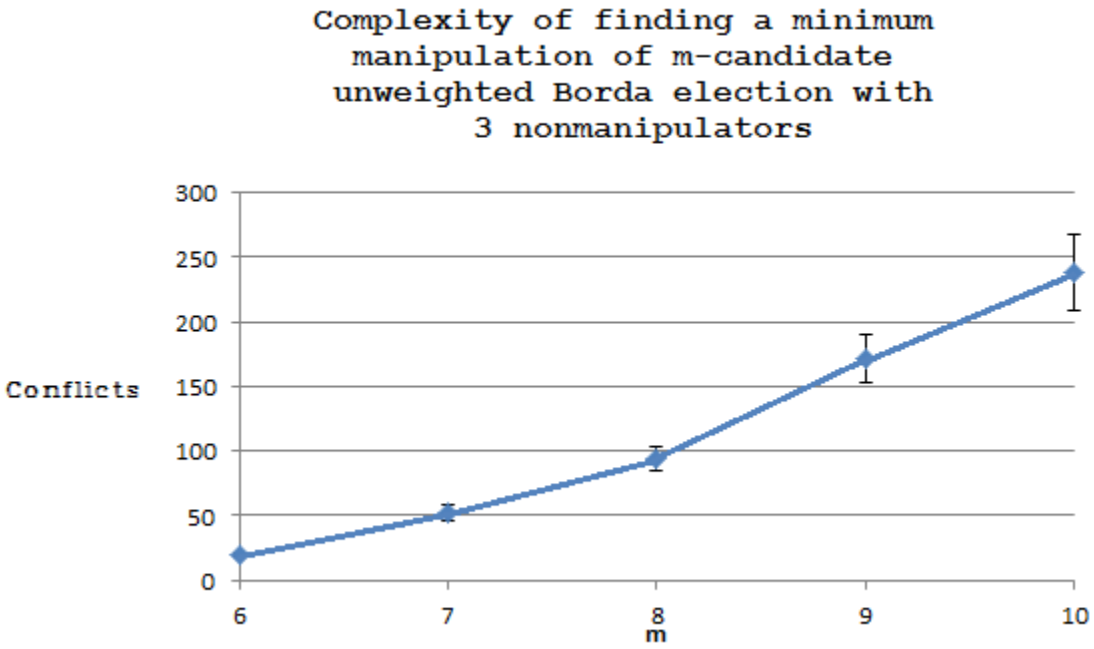
We find that this is not the case. In addition, we find that the same results appear even in the manipulation of unweighted Borda, despite our improvements on the reductions in [Con10] for symmetry. We plot the results below.



Even though the problem of manipulating unweighted Borda elections is NP-hard [DKNW11, BNW09], it is possible to approximate the minimum number of unweighted manipulators needed to affect the outcome of a Borda election by an additive factor of one manipulator

using a greedy algorithm [ZPR09]. For example, if seven manipulators are needed to make a distinguished candidate win an election, then the greedy algorithm will find a manipulation using either seven or eight manipulators. The greedy algorithm for manipulating unweighted Borda elections given in [ZPR09] operates by iteratively assigning c, the distinguished candidate, as the first candidate, and assigning the remaining positions by nonincreasing order of the current scores of the nondistinguished candidates.

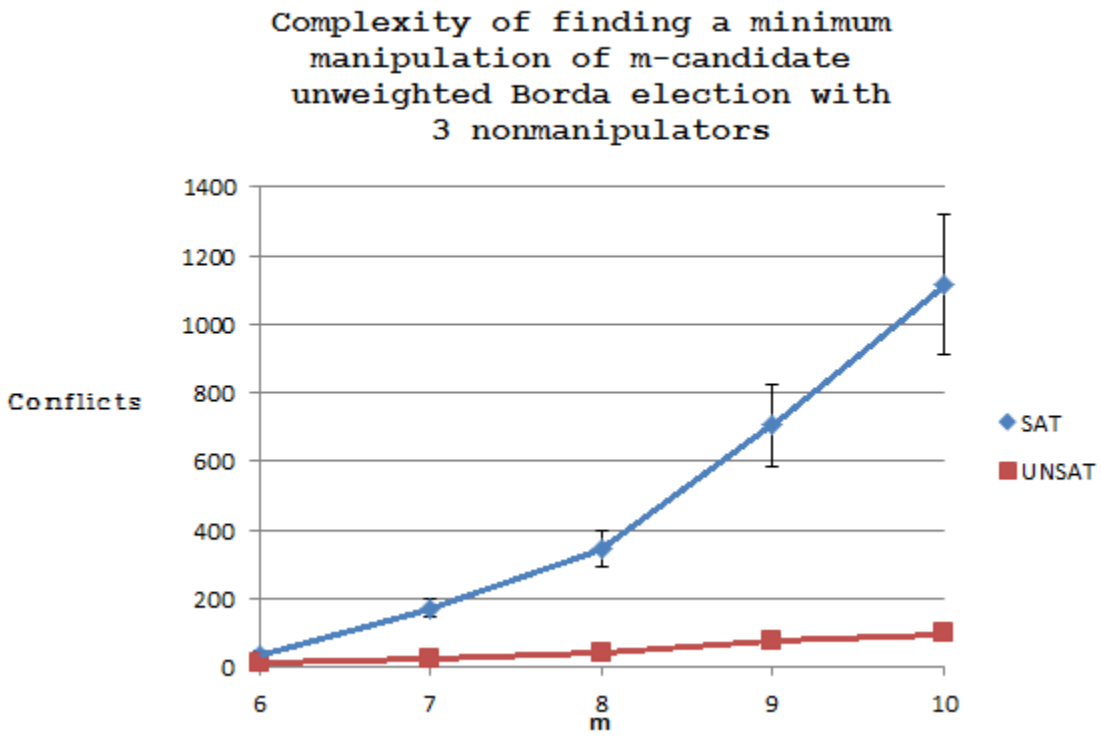
In our next test, we compute this greedy approximation, and attempt to manipulate the election with one fewer manipulator. Consequently, this allows us to determine the minimum number of candidates needed to manipulate the election. We evaluate the complexity of invoking Clasp in the SAT instance in doing so.



Surprisingly, the results above suggest that this encoding for manipulation appears to be most effective near the boundary of manipulability, showing a linear empirical complexity with respect to m, the number of candidates. This result of linear empirical mean complexity does not hold without symmetry breaking in the encoding (i.e., when using the encoding into 0-1 ILP for weighted elections).

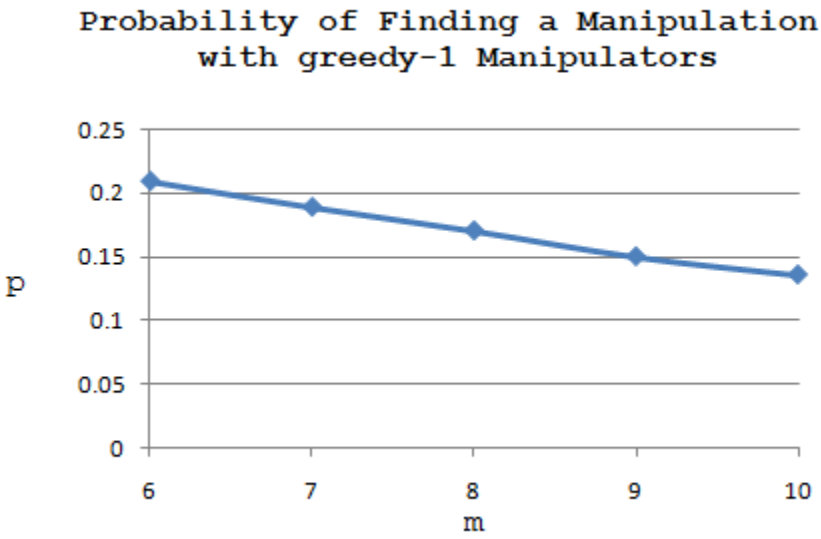
In our next test, we break the empirical data above into cases in which the formula is

satisfiable and unsatisfiable, i.e., whether or not the election can indeed be manipulated using one fewer voter than given by the greedy algorithm.

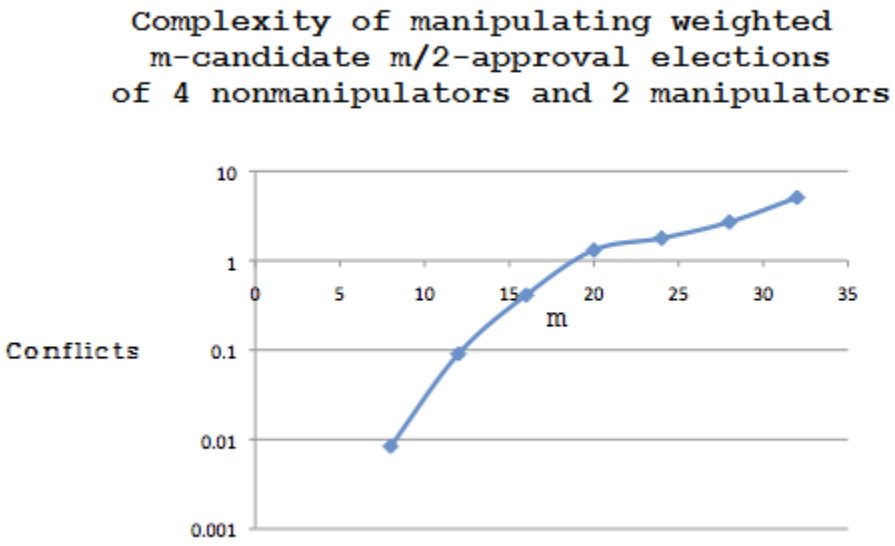


These results suggest that it is, in general, harder to show that no manipulation exists, for one fewer manipulator than given by the greedy algorithm in [ZPR09], than to find a manipulation of said size. This is in direct contrast to most instances of SAT and other NP-complete problems, in which unsatisfiability instances are generally harder.

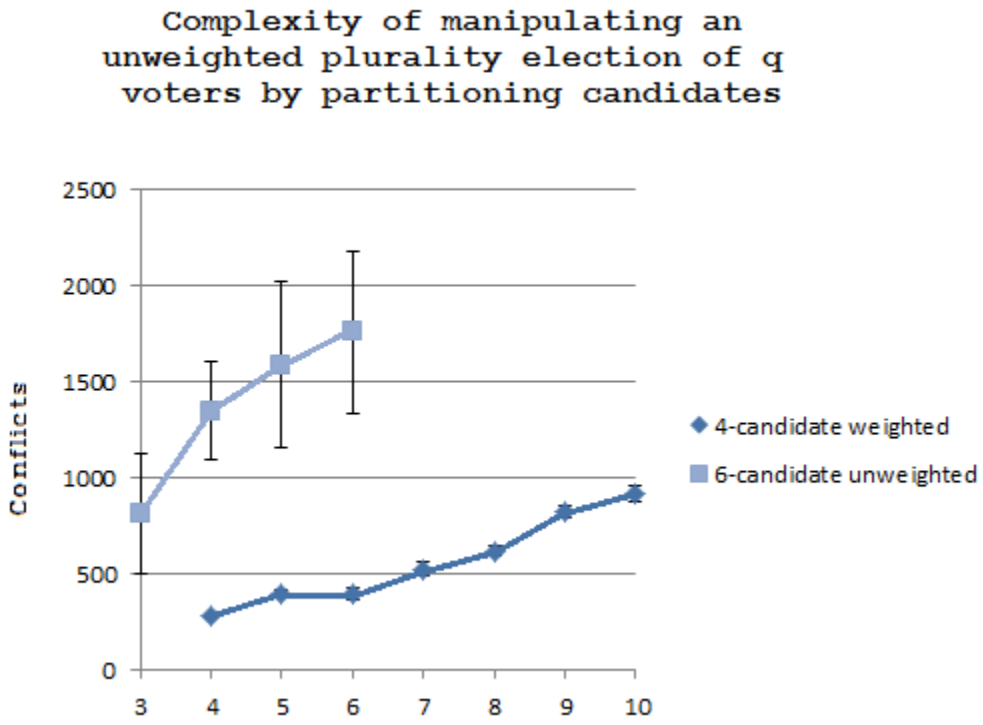
In the next diagram, we show that the probability of having a manipulation with one fewer manipulator than that given by the greedy algorithm is inversely correlated with the size of the candidate set, presumably tending toward zero. This suggests that the greedy algorithm is increasingly precise as the size of the candidate set is increased, and may partially be responsible for the low overall empirical complexity of this problem.



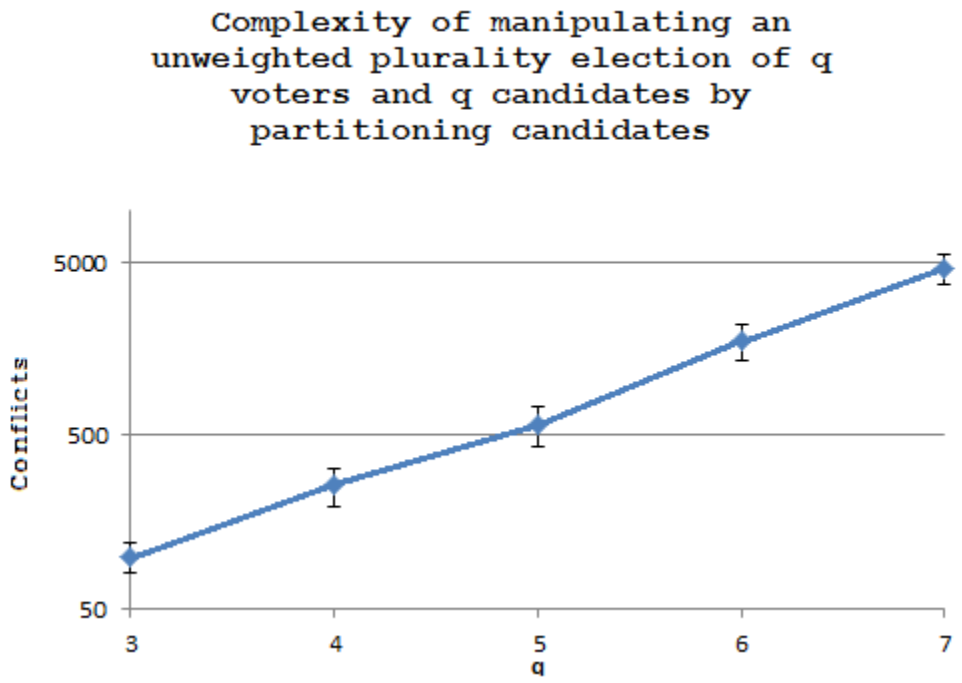
In our next test, we examine the family of scoring protocols, $\frac{m}{2}$-approval elections of m candidates, of four nonmanipulators and two manipulators. These results, unfortunately, also confirm the results using Partition algorithms in Chapter 4, that the Clasp algorithm on our reduction does not scale well with the size of the candidate set, even with a small voter set and a relatively simple election system.



Finally, we test our encoding on the problem of control by partitioning candidates, in both weighted and unweighted cases. This problem is polynomial-time computable in cases in which the size of the candidate set is fixed, and this is confirmed by our empirical testing. We plot two cases in the diagram below. Due to the large size of our encodings, we are able to run a limited number of rounds for some of the weighted cases, and this is reflected by the larger confidence intervals.



In our final test, we increase the size of the candidate set within a simple family of scoring protocols, that of plurality. We see that, like the cases of manipulation and bribery, the empirical complexity of the current state-of-the-art SAT solver on our encoding do not scale well with the size of the candidate set.



5.6 Discussion

These new results contrast some of our previous results in Chapter 4, that show that manipulation is not vulnerable to some natural extensions of algorithms for k-Way Partition in scoring protocols of more than three candidates. This may be due to a weakness in either the known algorithms for k-Way Partition, our extension thereof, or the reduction from the manipulation problem to that of k-Way Partition. As it is also shown in Chapter 4 that the problem of manipulation in weighted scoring protocols is closely related to that of Partition, we presume that some of these approaches could also improve our understanding of algorithms for Partition. Using a very similar SAT encoding, we also extend the results of Walsh, that hard instances of manipulation are exceedingly rare even within the phase transition of the problem, to the problem of weighted and unweighted bribery of scoring protocols of a fixed number of candidates.

The results also suggest that when the cardinality of the candidate set is not fixed, as in the case of families of scoring protocols, the election systems are not easy to manipulate, bribe, or control near the phase transition by the best-known solvers for SAT on obvious

encodings of these problems. Such an observation suggests that a large candidate set is a significant complexity barrier in many interesting problems in election systems, more so than a large voter set or a phase transition of the problem.

Some of our results in the manipulation in unweighted Borda elections also suggest an unusual behavior of complexity of the SAT solvers on our encodings of these problems. In particular, it is found empirically that it is often easier to solve this decision problem near the boundary of satisfiability: when the number of manipulators given is near the optimal manipulation, and that unsatisfiable instances are significantly, possibly asymptotically, easier than satisfiable instances in this case. These two observations are in direct contrast to many interesting instances of SAT and other NP-complete problems. Such properties appear to have allowed us to find the minimum manipulation fast empirically.

In all of the other data plotted above, we were interested in answering decision problems: whether or not an election could be manipulated or bribed. Approximation algorithms for finding the minimum manipulation or bribery also exist for many election systems, and a variation of this concept to find the minimum manipulation or bribery may possibly be applied to these systems, to investigate if such peculiarities in complexity also exist.

Chapter 6

Finding Hard Winners With SAT Solvers

An interesting winner criterion in elections is that of a Condorcet winner [Con85]. A Condorcet winner is a candidate who outperforms each other candidate in a pairwise election among the set of voters; it is preferred by a majority of voters against each other candidate. We give an example as follows.

Consider an election over three candidates, a, b, and c, and three voters, with preferences $v_1 = (a \succ b \succ c)$, $v_2 = (b \succ a \succ c)$, and $v_3 = (c \succ a \succ b)$. a is a Condorcet winner of this election, because two voters (a majority), v_1 and v_3, prefer a to b, and two voters, v_1 and v_2, prefer a to c. Note that despite an even plurality score in this election, the Condorcet winner is likely to be a candidate somewhat preferred by most of the voters.

Unfortunately, Condorcet himself [Con85] discovered that a Condorcet winner (or even a weak Condorcet winner, see Section 2.1.1 for definition of Condorcet and weak Condorcet winners) may not always exist, even in elections with as few as three candidates. The smallest counterexample showing that a Condorcet winner may not exist contains just three candidates and three voters. In an election of three candidates a, b, and c, and three voters, with preferences $v_1 = (a \succ b \succ c)$, $v_2 = (b \succ c \succ a)$, and $v_3 = (c \succ a \succ b)$, a majority of voters prefer a to b, b to c, but also c to a. Such an example is known as a majority cycle. The fact that aggregating the preferences of rational voters may lead to irrational pairwise preferences is known as the Condorcet paradox.

Because Condorcet winners, when they exist, have desirable properties, several election systems have been proposed that have the property of always choosing the Condorcet winner (or the set of weak Condorcet winners) when one exists. An election system with this

property is said to be Condorcet (or weak Condorcet) consistent. Many election systems, such as that of Borda, are not Condorcet consistent [Nan82].

In cases where a Condorcet winner does not exist, a Condorcet consistent voting system will generally attempt to find an approximate Condorcet winner by some metric. Two prominent examples of Condorcet consistent election systems include Dodgson [Dod76] and Young [You77, RSV03] elections (see Section 2.1.1 for distinctions between variations of Dodgson and Young elections).

In a Dodgson election, a candidate is considered "close to" being a Condorcet winner if it can be made a Condorcet winner by making transpositions of a small number of adjacent pairs of candidates in the preferences of the voters. In the three-voter election above showing the Condorcet paradox, a can be made a winner by transposing a and c in the preferences of v_2, giving a a pairwise outcome of two votes against either b or c. The Dodgson score of each candidate is the fewest number of transpositions of adjacent candidates one must perform in the preference orderings of the voters to make the candidate a Condorcet winner. Thus in the example above with the majority cycle, a has a Dodgson score of one, since one transposition suffices while zero transpositions cannot make a a Condorcet winner. The candidates with the lowest Dodgson score are thus the closest to being Condorcet winners in the election and win the Dodgson election. Note that a Condorcet winner has a Dodgson score of 0, as no transpositions are necessary, and is the unique Dodgson winner.

Another Condorcet consistent election system is that of the Young election. In this model, a candidate is "close to" being a Condorcet winner if it can be made a Condorcet winner of an election by deleting a small number of voters from the election. The Young score of a candidate c in an election is the fewest number of voters that must be deleted to make c a Condorcet winner of the election. The candidates with the lowest Young score win the Young election. A Condorcet winner also has a Young score of 0. (It should be noted that the Young score of a candidate c is also sometimes defined as the cardinality of the largest subset of voters under which c is a winner, making the Young winner the candidate(s) with the highest Young score. For consistency with the definitions of Dodgson score, we will not use this definition in the remainder of this thesis. This distinction has important implications to approximability of scoring candidates in Young elections, as approximations are expressed by ratios.)

Unfortunately, it is hard, at least in the worst case, to find either the scores of the candidates or to find the sets of winners in both of these systems. The scoring problem, of finding the score of a candidate in a Dodgson or Young election, is NP-hard [BTT89b, RSV03],

while the winner problem, of determining whether or not a candidate wins a Dodgson or Young election, is Θ_2^p-complete [HHR97, RSV03].

6.1 Previous Results

While the score and winner problems are known to be NP-hard and Θ_2^p-complete [BTT89b, HHR97, RSV03] respectively in both Dodgson and Young elections, several results have demonstrated probabilistic and randomized algorithms for finding approximate solutions, especially for the problems in Dodgson elections.

In [HH06], the greedy algorithm for finding the Dodgson score given in Section 2.8 is evaluated theoretically using probability theory and calculus, and it is found that when the number of voters is significantly greater than the number of candidates, the greedy algorithm nearly always finds the Dodgson scores and winners of an election. Furthermore, the algorithm is self-knowingly correct for almost all cases, meaning it can identify when it is correct. Recall from Section 2.8 that an analogous greedy algorithm also exists for computing Young scores. However, similar approximation results have not been shown, and in fact, it is shown in [CCF$^+$09] that the Young score, unlike the Dodgson score, cannot be approximated by any factor. Nonetheless, the analogous greedy algorithm for Young score gives an upper bound for the Young score of a candidate, which we will apply in our computation of Young scores and winners in a later section. Recall that we also give polynomial-time computable lower bounds for Dodgson and Young scores in Section 2.8.

Another notable result [CCF$^+$09] finds a probabilistic approximation algorithm of Dodgson Score using linear programming. A 0-1 Integer Linear Program encoding of the problem of Dodgson score given by [BTT89b] is approximated by relaxing the problem to a noninteger linear system and finding a rational solution. The approximation is computed using randomized rounding of the resulting solution of the linear system. The resulting randomized algorithm is contrasted with the deterministic greedy algorithm in [HH06]. It is found that the randomized algorithm has the property of being monotonic, a property not shared by the deterministic greedy algorithm. That is, if the position of a candidate is improved, the resulting randomized algorithm will, probabilistically, not give a worse score, whereas the deterministic greedy algorithm may. This property of nonmonotonicity in the greedy algorithm is undesirable because it defeats the properties of the Dodgson election, as one important property of Dodgson scores is monotonicity. It is further proposed that the property of monotonicity allows the randomized algorithm to behave like an actual election system. However,

it is also found that the problem of Young score cannot be approximated in the same way as that of Dodgson score [CCF$^+$09, CCF$^+$09], and analogous results have not been established.

Another result supporting the fact that finding the winners in Dodgson elections may be easier than that of Young elections is the parameterized complexity results given in [BGN10]. It is found that determining whether or not the Dodgson score of a candidate c in an election $E = (C, V)$ is at most q is polynomial-time computable when q is fixed. This is not the case for the Young election, in which this problem remains NP-hard with a fixed value of q, the target score.

Encodings of the problems of Dodgson and Young score to 0-1 Integer Linear Programming were first explicitly given in [BTT89b, CCF$^+$09]. In the encoding of Dodgson Score [BTT89b], the swaps to be made are encoded by defining variables to indicate how much c, the candidate for which we are computing the Dodgson score, is to be pushed up in the preference ordering of each voter. Without loss of generality, it may be assumed that the only swaps of adjacent candidates in voter preferences are to push c upward in the preference ordering. In the encoding for Young Score given in [CCF$^+$09], variables indicate the voters to be included in the subset of voters for which c is a Condorcet winner under. In both encodings, for Dodgson and Young scores, the constraints tell us that c is to beat each other candidate in a pairwise election of the resulting modified preference profile.

A related problem, a variation of bribery of Dodgson and Young election, has also been evaluated using ILP encodings [FHH09]. In the problems of DodgsonScore-succinct-bribery and YoungScore-succinct-bribery, the goal is to bribe the voters of the election to improve the score of a candidate c in a Dodgson or Young election, changing the preferences of at most q voters and improving the score of the c by at least some difference d. The voter preferences are represented succinctly. In the reduction given by [FHH09], the variables in the encoding count the number of voters of each preference that are to be affected, using succinct representation. It is shown that because the number of variables in the encoding is bounded for a fixed candidate set size m, that this problem is polynomial-time computable in this setting (see the ILP result in [Len83]). However, it is pointed out in [FHH09] that the more familiar bribery problem, of bribing to elect a preferred candidate, is not likely to be easily encoded as a 0-1 ILP instance. We will touch on possible solutions for the similar problem of manipulation in a later section.

6.2 Dodgson and Young Score Problems and 0-1 Integer Linear Programming

In this section, we will be reviewing the encodings of Dodgson and Young Score given by [BTT89b, CCF+09], and making some adjustments to achieve symmetry breaking.

As we have shown in Chapter 5 for cases of manipulation problems, symmetry may also exist in the problem of finding scores in Dodgson or Young elections, and may lead to inefficiency in solving the resulting encodings. For example, voters may have identical preferences. Because voters are unweighted in the Dodgson and Young elections we are interested in, it suffices to count the number of voters to be affected. In the case of computing the Dodgson score of c, for example, it suffices to count the number of voters with each preference ordering, such as $a \succ b \succ c$, in which c is to be pushed up by each number of places, either one or two in this case. The exact set of voters with preference $a \succ b \succ c$ in which c is to be improved by one place do not need to be enumerated, as they are interchangeable. In Young elections, it suffices to count the number of voters with preference each ordering, such as $a \succ b \succ c$, to include in the subset of voters for which c is a Condorcet winner. The choice of voters of preference $a \succ b \succ c$ to include need not be enumerated. The encodings in [BTT89b, CCF+09] may be inefficient in elections with many similar voters, because the encoding may cause the SAT solver to enumerate a large number of similar unsuccessful modifications of the election.

More specifically, we may break symmetry by representing the preference profile succinctly and counting the number of voters of each preference affected in the modification of the preference profile. In both cases, the symmetry breaking prevents one from representing modifications having the same effect with more than one instantiation of the variables, which may cause potential SAT solvers to "thrash." Recall the discussion of how depth-first-search SAT solvers operate and the benefits of symmetry breaking in Chapters 2 and 5. The number of symmetric alterations to the preference ordering grows exponentially with respect to the size of the voter set, as voters with similar preferences and that are affected similarly by the alteration may be permuted.

Theorem 6.2.1 *It is possible to make use of succinct preferences to break symmetry in the* 0-1 *ILP encoding of Dodgson Score given by [BTT89b].*

Proof: Consider an election of m candidates, $C = \{c_1, \ldots, c_m\}$ and n voters, $V = \{v_1, \ldots, v_n\}$. Suppose there are r distinct preference orderings among the voters, call them $P_1, \ldots, P_r$. For

each preference ordering P, let $1 \leq N(P) \leq m$ be the number of voters having such an ordering. We wish determine whether the Dodgson score of candidate c_1 is at most q, an NP-complete problem.

We pre-compute the following variables. For $c' \neq c_1$, let the deficit of c_1 in relation to c', or $\texttt{def}(c')$, be the number of additional voters that must rank c_1 above c' so that c_1 wins in a pairwise election against c'. For example, if five voters rank c' above c_1 and two vice versa, then $\texttt{def}(c') = 2$, since switching the order of c and c' in the preferences of two of these voters makes four voters rank c_1 above c' and three vice versa. If c_1 beats c' in a pairwise election initially, then $\texttt{def}(c') = 0$.

Further, we precompute $e_{P,l,c'}$ to be 1 if pushing c in the preference ordering P by l spaces gives c a lead against c'. For each preference P and candidate c, let $1 \leq \texttt{pos}(P, c) \leq m$ denote the position of c in the ordering of P.

It is stated in [CCF+09] and easy to see that without loss of generality, it can be assumed that the only swaps of adjacent candidates we will make is to push the candidate c_1 upward. We define the variables of a 0-1 ILP problem as follows. For each $1 \leq i \leq r$, $1 \leq j \leq N(P_i)$, and $0 \leq l \leq \texttt{pos}(P_i, c_1) - 1$, let the variable $x_{i,j,l} = 1$ iff we are to push c_1 in exactly j voters of preference P_i by exactly l places upward in the preference ordering. The constraint $\sum_{i,j,l} jlx_{i,j,l} \leq q$ tells us we are to make at most q swaps. For each $1 \leq i \leq r$ and $1 \leq l \leq pos(P_i, c) - 1$, we must push some number (possibly zero), j, of voters of preference P_i by l places. Thus, $\sum_j x_{i,j,l} \leq 1$. As there are $N(P_i)$ voters of preference P_i, for each $1 \leq i \leq r$, $\sum_{j,l} jx_{i,j,l} \leq N(P_i)$, indicating that a total of at most $N(P_i)$ voters of preference P_i will be affected by the swapping. For each candidate $c' \neq c_1$, c_1 is pushed ahead of c' in $\sum_{i,j,l} je_{P_i,l,c'}x_{i,j,l}$ voter's preferences. This quantity must be at least $\texttt{def}(c')$, ensuring the victory of c in a pairwise election against c'. This inequality differs from that in [BTT89b] in that the swapping of c_1 in more than one voters' preferences may be represented by one variable, $x_{i,j,l}$ variable if they occur in voters with the same preferences and the swaps are identical. This eliminates cases of symmetry in which voters with identical initial preferences may be permuted in how the preferences are to be affected.

It is also possible to combine voters with nonidentical preferences in some cases. Recall that c_1 is to only be pushed upward, and that only pairwise elections of c_1 against each other candidate are of importance. Thus, if some voters have preference ordering $c_2 \succ c_3 \succ c_1 \succ c_4 \succ c_5$ while some have $c_2 \succ c_3 \succ c_1 \succ c_5 \succ c_4$, it suffices to count the total number of both

types of preferences in which c_1 is to be pushed upward by each number of positions, in this case, one or two.

This adjustment is most likely to improve the complexity of the encodings when c_1, the candidate in which we are computing the Dodgson score for, is somewhat preferred by most of the voters, as few candidates are ahead of c_1 in the preferences of most voters. Thus, this is likely to further simplify the process of finding the Dodgson score of a candidate with a relatively low Dodgson score. As we will see, this will be important in our algorithm for finding the Dodgson winner in the next section.

There are $\frac{(m-1)!}{(m-i)!}$ distinguishable preferences in which c_1 is ranked in the i^{th} position, as only the ordering of the candidates ahead of c_1 is of importance, and thus a total of $\sum_{1 \leq i \leq m} \frac{(m-1)!}{(m-i)!}$ distinguishable preferences. ■

Theorem 6.2.2 *It is also possible to break symmetry using succinct preferences in the encoding for Young Score in [CCF+09].*

Proof: Consider an election of m candidates, $C = \{c_1, \ldots, c_m\}$ and n voters, $V = \{v_1, \ldots, v_n\}$. Suppose there are r distinct preference orderings among the voters, call them $P_1, \ldots, P_r$. For each preference ordering P, let $N(P) \geq 1$ be the number of voters having such an ordering. In this problem, we wish determine whether the Young score of candidate c_1 is at most q, an NP-complete problem.

We pre-compute the following variables. For each preference P and candidate $c' \neq c_1$, we define $e^P_{c'} \in \{-1, 1\}$ to be 1 if c_1 beats c' in preference P and -1 otherwise.

For each $1 \leq i \leq r$, $1 \leq j \leq N(P_i)$, let the variable $x_{i,j} = 1$ iff we are to include exactly j voters having preference ordering P_i in our set of voters in which c_1 is a Condorcet winner. For each $1 \leq i \leq r$, we must include some number (possibly zero) of voters of preference P_i. Thus, for all $1 \leq i \leq r$, $\sum_j x_{i,j} \leq 1$.

The linear sum $\sum_{i,j} j x_{i,j}$ tells us how many voters are included in this set, and thus should be at least $m - q$. For each $c' \neq c_1$, it must be the case that $\sum_{i,j} j x_{i,j} e^{P_i}_{c'} \geq 1$, to ensure that c_1 wins a pairwise election against c' among the set of voters to be included. This inequality differs from [CCF+09] in that we must count the number of voters of each preferences to be affected, as the election is represented succinctly.

We observe it suffices to only distinguish voters based on their pairwise preferences between c_1 and each $c' \neq c_1$. There are 2^{m-1} distinguishable preferences. Like the adjustment

in the encoding for Dodgson Score, this observation is also likely to improve the complexity of finding the Young score of a candidate which is somewhat preferred by most of the voters, as the set of voters beating c_1 will be relatively small for each voter, giving us few distinguishable preferences. This is also important for the algorithm to be given in the next section for finding Young winners in an election. ■

To find the Dodgson or Young score of a candidate in an election, one may use binary search on the parameter q, finding solutions to the reductions as given above, starting with the trivial lower bound and the upper bound given by the greedy algorithm [HH06] (see Section 2.8). As we will see in the next section, to find the Dodgson and Young winners of an election, it is not always necessary to compute the exact Dodgson and Young scores of each candidate.

6.3 Quering the Dodgson and Young Score Problem

In this section, we demonstrate how to query the NP-complete problems of Dodgson and Young Score to find the winners of an election or to rank the candidates in order by their Dodgson or Young score. In a later section, we will compare and contrast the problems of scoring a candidate, finding the winners, and ranking the candidates for both systems.

To prove that c is a winner of a Dodgson (Young) election, it suffices to prove that the Dodgson (Young) score of c is at most that of each candidate other than c. It is not necessary to compute the score of each candidate of the election. It is also possible to rank the candidates by Dodgson or Young score without computing the exact score of each candidate.

We give an example as follows. Consider a Dodgson election of four candidates, $C = \{c_1, c_2, c_3, c_4\}$. We compute a lower and upper bound of the Dodgson score (see Section 2.8), giving us bounds for the Dodgson scores of each candidate as follows: $3 \leq s(c_1) \leq 9$, $8 \leq s(c_2) \leq 11$, $2 \leq s(c_3) \leq 13$, and $6 \leq s(c_4) \leq 7$.

The candidate c_3 has the lowest lower bound, and the Dodgson score of c_3 is between 2 and 13 inclusive. We use binary search, starting with these bounds, to find the Dodgson score of c_3. Suppose it is found that the Dodgson score of c_3 is 5. Clearly, neither c_2 nor c_4 can win this election, so we will not solve the encodings of the Dodgson score problem for either of these candidates. We see that the Dodgson score of c_1 is between 3 and 9 inclusive. If the score is at least 6, it is not necessary to know exactly the score, as c_1 cannot win this election. Similarly, if the Dodgson score of c_1 is at most 4, then c_1 will win this election.

Thus, it suffices to perform a binary search on the Dodgson score of c_1 between 4 and 6 inclusive. This will either tell us that the Dodgson score of c_1 is exactly 5, or show that it is at least 6 or at most 4. If the score is found to be at most 4, then c_1 is the unique winner of this election. If it is 5, then c_1 and c_3 are the winners. Otherwise, if the score is found to be at least 6, then c_3 is the unique winner. In some cases the upper and lower bounds of the Dodgson (Young) score of each candidate are sufficient for establishing the winners of the election, and the SAT solver need not be queried at all.

Suppose that, instead, we wish to rank the set of candidates based on their score. Consider the example from above, in which bounds for the Dodgson scores of candidates c_1, c_2, c_3, and c_4 are found as follows: $3 \leq s(c_1) \leq 9$, $8 \leq s(c_2) \leq 11$, $2 \leq s(c_3) \leq 13$, and $6 \leq s(c_4) \leq 7$. As mentioned earlier, we sort the candidates by the lower bound of the score, in this case, evaluating c_3 first. The Dodgson score of c_3 is between 2 and 13 inclusive. Suppose as above that the score is found using binary search to be 5. The candidate with the next lowest lower bound is c_1, of which the Dodgson score is known to be between 3 and 9 inclusive. Because the score of c_3 is found to be 5, and the scores of c_4 and c_2 are known to be at least 6 and 8 respectively, if the score of c_1 is either 3 or 4, it does not suffice to know the exact score of c_1. Thus, it suffices to search for the score of c_1 from between 4 and 9 inclusive. Let us suppose the score is found to be 7. We next find the score of c_4 from between 6 and 7 inclusive. Suppose it is found to be 7. Since the lower bound of the score of c_2 is 8, and the scores of c_3, c_1, and c_4 are found to be 5, 7, and 7 respectively, c_2 clearly has a higher score than each of the other candidates. No queries is needed for the score of c_2. This tells us that $s(c_3) \leq s(c_1) = s(c_4) \leq s(c_2)$.

The winner problem example demonstrates that finding candidates with low scores early prunes the search significantly, by eliminating candidates which cannot beat the given score. Also note that the candidates with high lower and upper bounds (c_2 and c_4 in this example) are likely to not be evaluated. This observation is notable, because a previous result [BGN10] has shown that finding the Dodgson score of candidates with low scores is easy. We conjecture that even if finding the score of some candidates or ranking the candidates in a Dodgson election is hard, the problem of finding the set of winners may possibly not inherit the hardness properties of the scoring and ranking problems, as the scores of some of the worst candidates in the election, in which the scoring problem is hard, may not be needed. As we will show later in this chapter, our experimental results support this conjecture.

For the reasons above, we wish to start our search with candidates likely to have low Dodgson scores. Experimentally, we have found that evaluating the candidates with the

lowest lower bounds first has performed better than evaluating the candidates with the lowest upper bounds first.

The algorithm for finding the set of winners of a Dodgson election can be described as follows. We precompute the lower and upper bounds of the Dodgson score of each candidate (recall the greedy algorithm and obvious lower bound in Section 2.8), and sort the candidates in nondecreasing order by the lower bound of their Dodgson score. For each candidate, it suffices to find the score up to the current best score, or show that it is worse than such. At the lower bound of the search, it suffices to show that it will beat both the current best score and each lower bound of the scores of the remaining candidates, showing it is the unique winner of the election. We find the score between these bounds using binary search. It is not always necessary to find the exact score of a candidate.

The pseudocode for the algorithm described in the example above can be given as follows.

DodgsonWinners$(E = (C, V))$

for each $c \in C$

Computer a lower and upper bound for the Dodgson score of c as follows:

$\texttt{Lower}(c) \leftarrow \sum_{c' \neq c} \texttt{def}(c, c')$, the total deficit of candidate c against each other candidate.

$\texttt{Upper}(c) \leftarrow$ greedy approximate Dodgson score of c.

Sort the candidates $C = \{c_1, \ldots, c_m\}$ by the lower bound of the Dodgson score in nondecreasing order (i.e., such that $\texttt{Lower}(c_1) \leq \cdots \leq \texttt{Lower}(c_m)$.)

if $(\texttt{Upper}(c_1) < \texttt{Lower}(c_2))$ c_1 is the only Dodgson winner.

$\texttt{BestScore} \leftarrow \texttt{Upper}(c_1)$; `BestScore` will contain the best Dodgson score found thus far.

for i **from** 1 **to** m

Compute bounds for searching for the Dodgson score of c_i as follows:

$q_{\min} \leftarrow \max(\texttt{Lower}(c_i), \min(\texttt{BestScore}, \texttt{Lower}(c_{i+1})) - 1)$

$q_{\max} \leftarrow \min(\texttt{Upper}(c_i), \texttt{BestScore} + 1)$

Using binary search, either find the Dodgson score of c_i if it is between $q_{\min} + 1$ and $q_{\max} - 1$ inclusive, or show that it is at least $q_{\max}$ or at most $q_{\min}$, and store the result in `Score` as follows:

$\texttt{Score} \leftarrow q_{\min}$ if the Dodgson score of c_i is at most $q_{\min}$.

$\texttt{Score} \leftarrow q_{\max}$ if the Dodgson score of c_i is at least $q_{\max}$.

$\texttt{Score} \leftarrow$ the Dodgson score of c_i otherwise.

if $(\texttt{Score} = \texttt{BestScore})$

Mark c_i as a winner.

if ($\texttt{Score} < \texttt{BestScore}$)

Mark c_i as a winner and unmark $c_1, \ldots, c_{i-1}$ as winners.

$\texttt{BestScore} \leftarrow \texttt{Score}$

Return the set of winners.

The pseudocode for finding Young winners is similar, with the exception that instead of using the total deficit of a candidate, we make use of the maximum deficit as a lower bound for the Young score (see Section 2.8).

In the next problem of interest, we wish to rank the set of candidates of an election in order by their Dodgson score. As before, we sort the candidates in nondecreasing order of the lower bound of the score, i.e., $\texttt{Lower}(c_1) \leq \cdots \leq \texttt{Lower}(c_m)$, and evaluate the score of the candidates in that order.

Consider the candidate c_i. If it is found that $\texttt{Upper}(c_i) < \texttt{Lower}(c_{i+1})$, then c_i will lose to $c_{i+1}, \ldots, c_m$, and it suffices to search for the score of c_i up to $\max(\texttt{Score}(c_1), \ldots, \texttt{Score}(c_{i-1}))+1$, allowing us to distinguish the position of c_i relative to $c_1, \ldots, c_{i-1}$.

The lowest score in the election possible at this point is given by $\min(\texttt{Score}(c_1), \ldots, \texttt{Score}(c_{i-1}), \texttt{Lower}(c_{i+1}))$ and it suffices to search for the score of c_i down to a minimum of $\min(\texttt{Score}(c_1), \ldots, \texttt{Score}(c_{i-1}), \texttt{Lower}(c_{i+1})) - 1$.

The pseudocode for ranking the candidates by Dodgson score can be given as follows.

DodgsonRanking($E = (C, V)$)

for each $c \in C$

Computer a lower and upper bound for the Dodgson score of c as follows:

$\texttt{Lower}(c) \leftarrow \sum_{c' \neq c} \texttt{def}(c, c')$, the total deficit of candidate c against each other candidate.

$\texttt{Upper}(c) \leftarrow$ greedy approximate Dodgson score of c.

Sort the candidates $C = \{c_1, \ldots, c_m\}$ by the lower bound of the Dodgson score in nondecreasing order (i.e., such that $\texttt{Lower}(c_1) \leq \cdots \leq \texttt{Lower}(c_m)$.)

for i **from** 1 **to** m

Compute bounds for searching for the Dodgson score of c_i as follows:

$q_{\min} \leftarrow \max(\texttt{Lower}(c_i), \min(\texttt{Score}(c_1), \ldots, \texttt{Score}(c_{i-1}), \texttt{Lower}(c_{i+1})) - 1)$

`if` $\texttt{Upper}(c_i) < \texttt{Lower}(c_{i+1})$

$q_{\max} \leftarrow \min(\texttt{Upper}(c_i), \max(\texttt{Score}(c_1), \ldots, \texttt{Score}(c_{i-1})) + 1)$

`else`

$q_{\max} \leftarrow \texttt{Upper}(c_i)$

Using binary search, either find the Dodgson score of c_i if it is between $q_{\min} + 1$ and $q_{\max} - 1$ inclusive, or show that it is at least $q_{\max}$ or at most $q_{\min}$, and store the result in $\texttt{Score}$ as follows:

$\texttt{Score}(\texttt{c}_\texttt{i}) \leftarrow q_{\min}$ if the Dodgson score of c_i is at most $q_{\min}$.

$\texttt{Score}(\texttt{c}_\texttt{i}) \leftarrow q_{\max}$ if the Dodgson score of c_i is at least $q_{\max}$.

$\texttt{Score}(\texttt{c}_\texttt{i}) \leftarrow$ the Dodgson score of c_i otherwise.

$\texttt{Score}(\texttt{c}_\texttt{1}), \ldots, \texttt{Score}(\texttt{c}_\texttt{m})$ will reflect the order of the Dodgson score of the candidates.

This algorithm can also be used to rank the Young scores of the candidates of an election.

In the empirical results in the next section, we demonstrate that the three related problem of finding a Dodgson or Young score, ranking the candidates by score, and finding the winners, have vastly different complexity behaviors, for the reasons described above. Most notably, that it is not necessary to zero in on the exact scores of each candidate in order for one to determine the winners or to rank the candidates. As a result, some of the results of the winner problem contrast those of the scoring problem in earlier results, especially in Young elections. Also, potentially due to the relative easiness in scoring candidates with a low Dodgson or Young score [BGN10], we also find interesting disparities in the problems of finding the set of winners and of ranking the candidates.

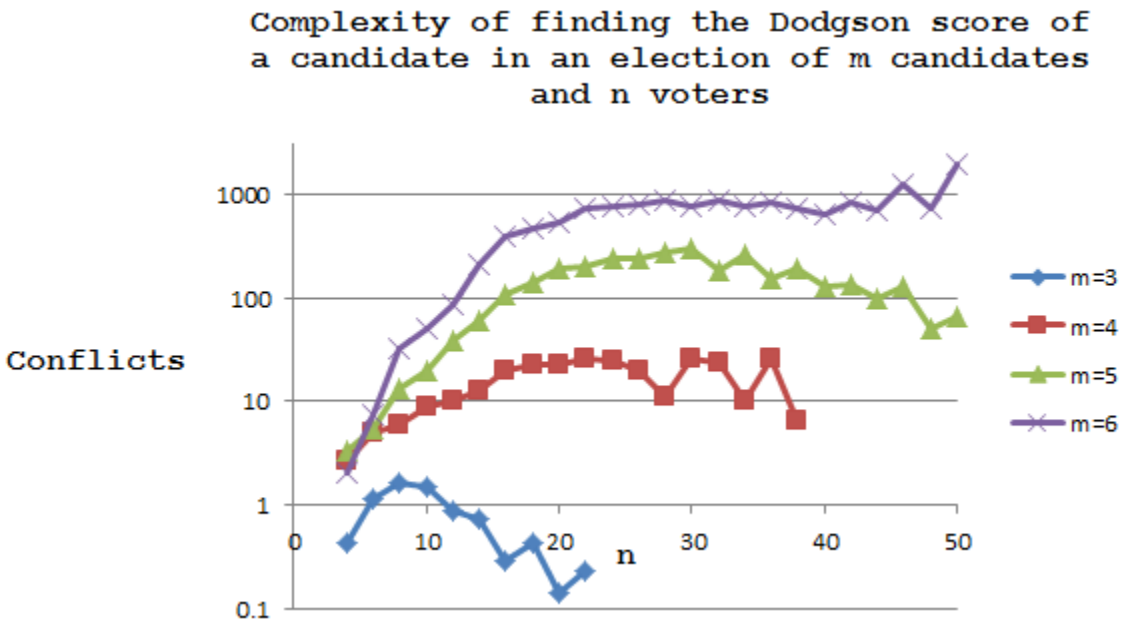
6.4 Experimental Results

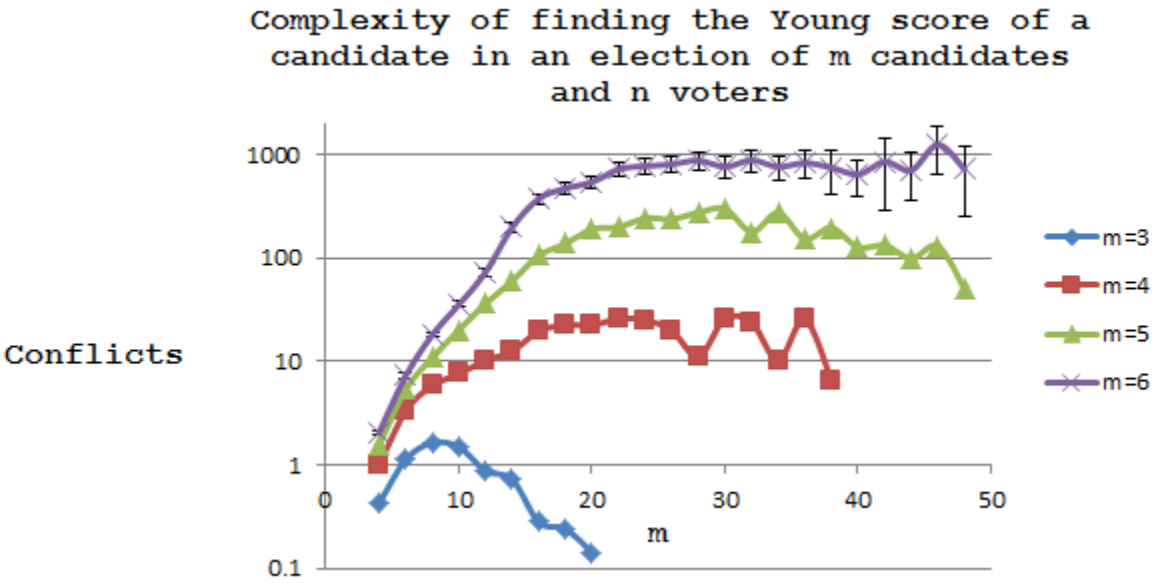
We experimentally evaluate the feasibility of finding scores, finding winners, and ranking the candidates in Dodgson and Young elections using the above 0-1 ILP encodings and algorithms. In doing so, we wish to make the first attempt at generalizing the concept of a phase transition from that of an NP-complete problem to that of a Θ_2^p-complete problem, and gain a better understanding of the frequency of hard instances of problems of this class. Recall that in a case of an NP-complete problem a phase transition occurs when the probability of a solution abruptly changes from almost one to almost zero as constraints are introduced beyond a threshold, and it has been found empirically in multiple problems that instances near the phase transition challenge many algorithms. We also wish to gain an understanding of the similarities and differences of the scoring and winner problems for both of these systems.

Let m be the number of candidates and n the number of voters of an election. We sample the set of elections of these parameters as follows. The distribution of each random voter's preferences are independent, and a shuffle algorithm is used to generate the preference of each voter. Recall from Chapter 5 that a shuffle algorithm outputs a permutation of m items such that the probability of each permutation of each permutation is exactly $\frac{1}{m!}$. The shuffle algorithm in use is described in more detail in Chapter 5.

Using a shuffle algorithm to generate random preference orderings independently for each voter is consistent with the definition of random voters in [Wal09] in the case of veto elections of three candidates. All of the elections tested in this chapter are unweighted.

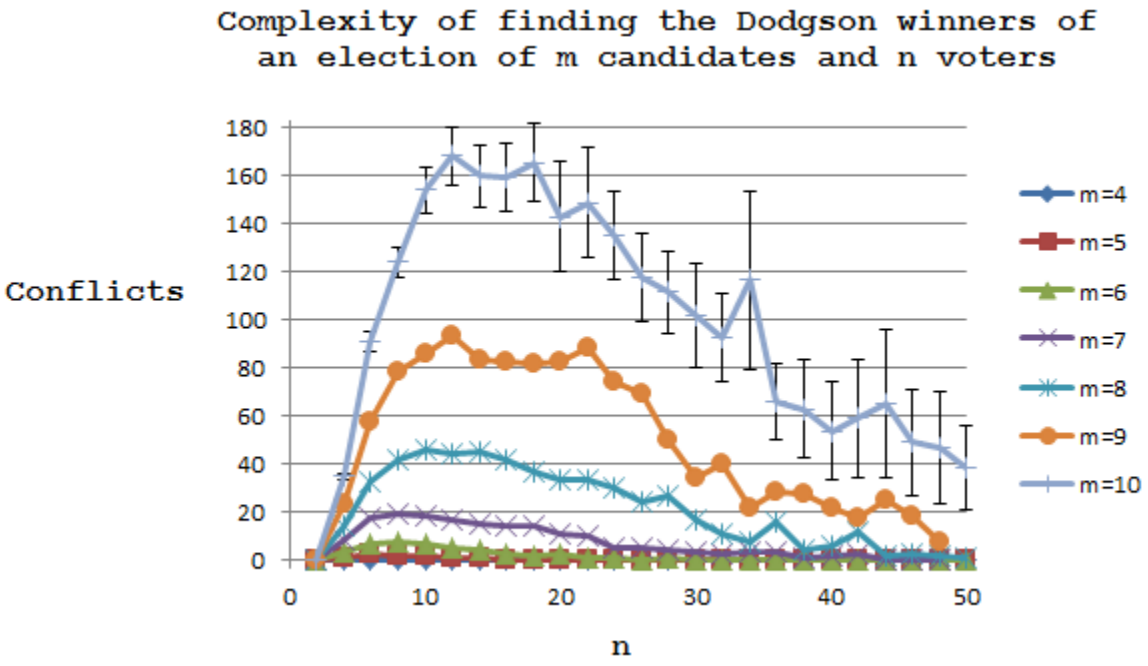
In our first test, we evaluate the empirical complexity of finding the Dodgson or Young score of a random candidate in 10,000 random elections. We do so by finding the lower and upper bounds of the score of the candidate as described in Section 2.8, and using binary search by invoking the SAT solver on the encodings above. We count the average total number of conflicts encountered by the algorithm of SAT. Recall from Chapter 5 that a conflict occurs in a SAT solver algorithm when the algorithm backtracks, and is consistent with the definition of branches in the testing in [Wal09] as well as the cases in Chapter 4. Finding the Dodgson or Young score of a candidate in an election, unlike the problems in Chapter 5, may involve invoking the SAT solver more than once or not at all (if the lower and upper bounds agree). In the latter case, there are 0 conflicts in total.





These results suggest the existence of a phase transition, in which the empirical complexity of the problem levels off as the number of voters is increased beyond a threshold relative to the number of voters. Unfortunately, unlike the results in [Wal09], the peak complexity of finding the Dodgson or Young score of an election appears to grow exponentially with respect to the size of the voter set under our encoding. In our next experimental test, we show that this weakness appears limited to the problem of finding the scores of the candidates in an election, and that the problem of finding the Dodgson and Young winners of an election do not appear to inherit this hardness property. This is due to the pruning techniques that eliminate the need to find the score of some of the candidates in an election in which we wish to find the winners.

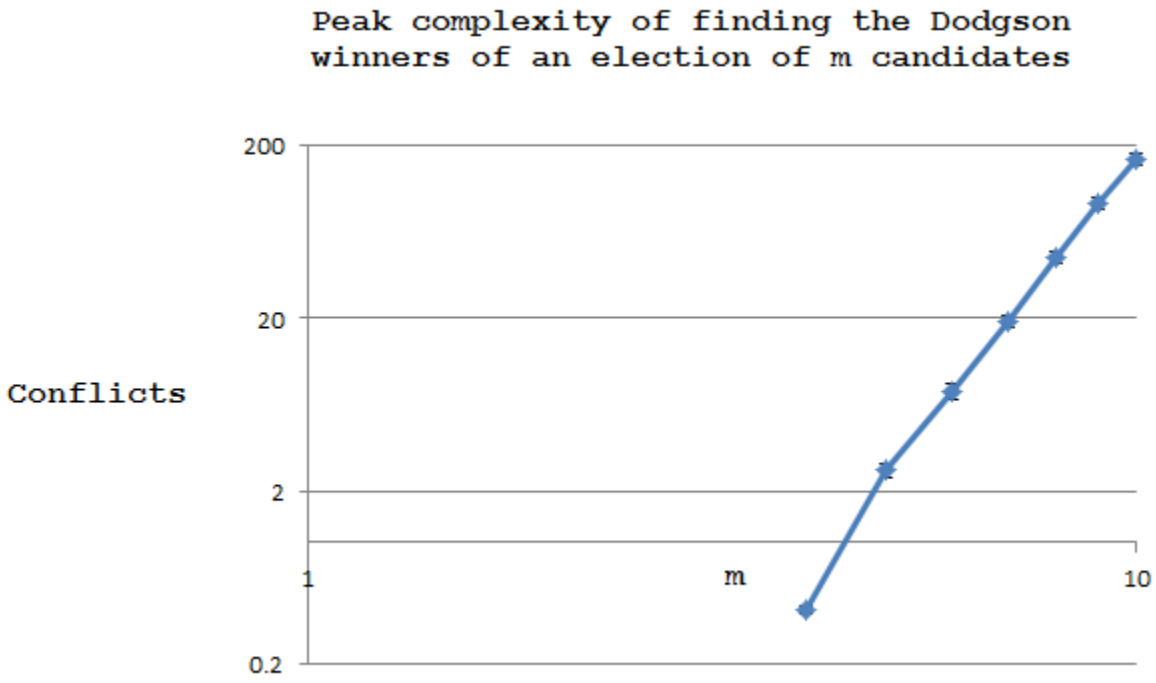
The empirical complexity of finding the Dodgson winner using the algorithm described above for elections of $4 \leq m \leq 10$ candidates is shown below. We measure this by evaluating the total number of conflicts encountered by the algorithm of SAT on all of the instances evaluated by SAT for each election. We evaluate 10,000 random elections, and also plot the 95% confidence interval of this complexity for $m = 10$ to ensure statistical significance.



Recall that in [HH06] it is found that the greedy algorithm of Dodgson Score is almost always self-knowingly correct when the number of voters is significantly larger than the number of candidates. This is confirmed in the results in that the complexity of finding Dodgson winners drops significantly as the number of voters is increased. This is likely because the lower and upper bounds of the Dodgson score are likely to converge for a large voter set, reducing the number of queries to the SAT solver.

We also find that the number of voters in which the empirical complexity of finding the Dodgson winners reaches a peak is roughly independent of the number of candidates, with hard problems roughly lying in the region where there are about 12 voters. However, the area of the number of voters in which hard instance occur appears to widen as m is increased.

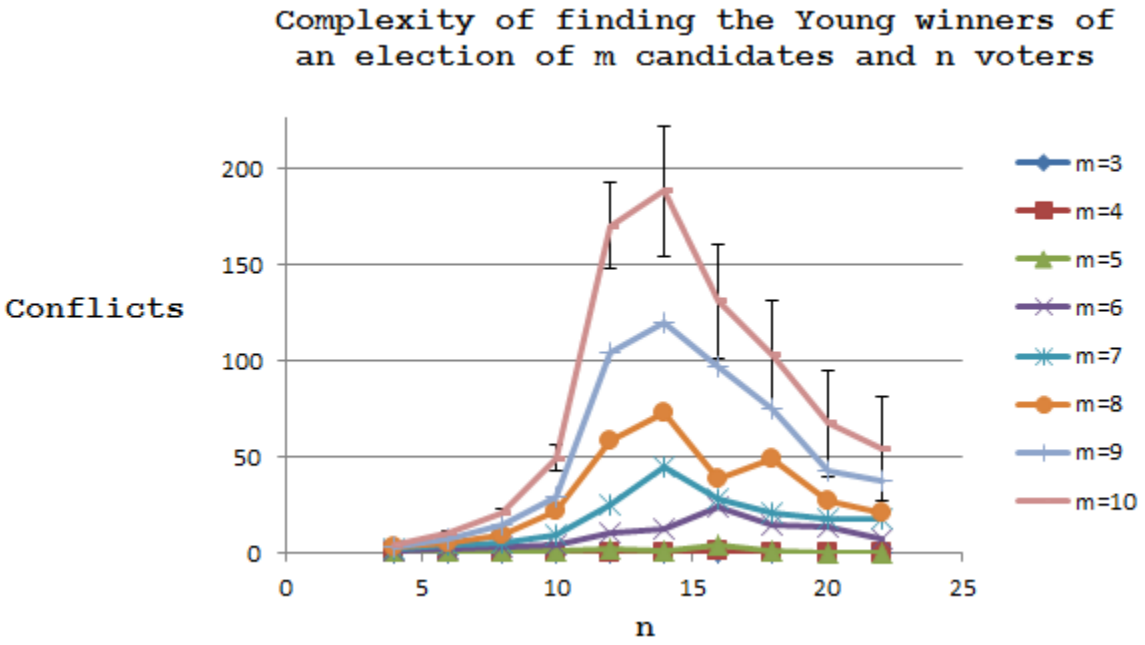
In the following diagram, we observe that the peak in complexity of finding Dodgson winners in elections of m candidates appears to be polynomial with respect to m. This is in contrast to the results of finding the exact Dodgson score. We make this observation more apparent by plotting the complexity with a log scale on each axis.

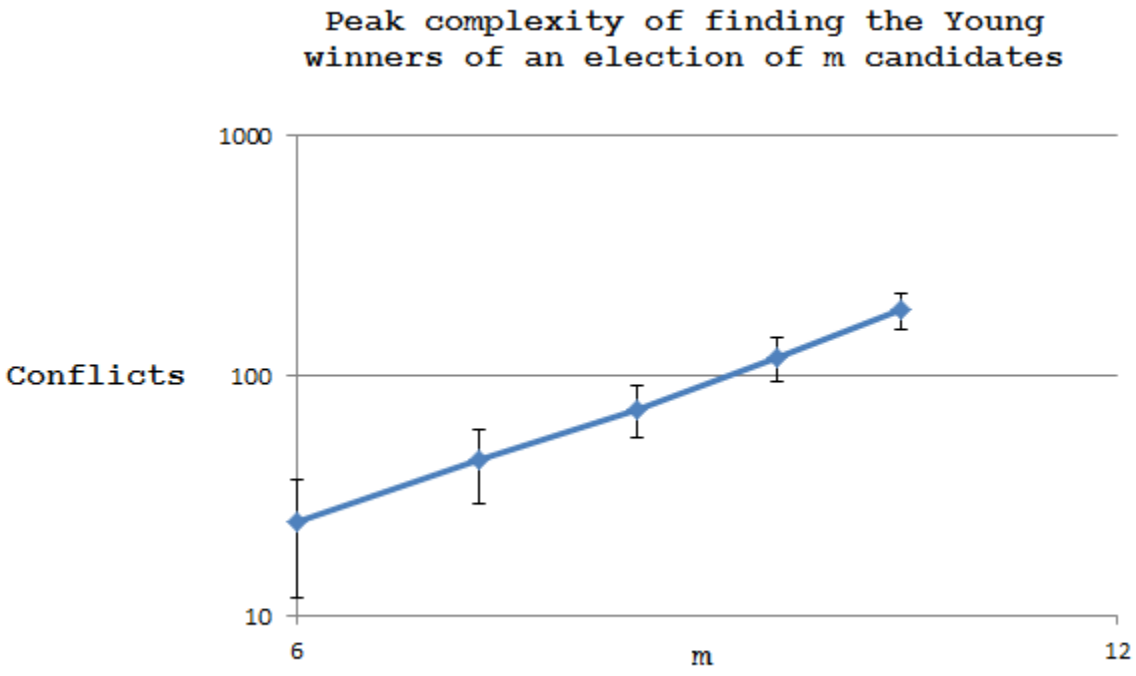


A good fit appears to show that this complexity is $\Theta(m^3)$, which is roughly the number of variables in the instance of the 0-1 ILP encoding. This means that, like the problem of manipulation studied by Walsh [Wal09], instances of elections in which finding the Dodgson winners is hard are exponentially rare.

This is in contrast with the seemingly easier problem of finding the Dodgson score of a single candidate, using the same underlying SAT encoding. This surprising result is due to the fact that it is not always necessary to find the Dodgson score of each candidate to determine the set of winners of the election. Additionally, we are only interested in the winner, which is likely to have a low Dodgson score. A result in [BGN10] shows that finding the Dodgson score of a candidate of a low score is easy. Recall from our algorithm that finding a candidate with a low Dodgson score allows us to ignore candidates with a lower bound exceeding this score, potentially avoiding expensive computations of Dodgson scores for candidates having a large score. Furthermore, empirical testing also shows that it is better, although apparently not asymptotically, to sort the candidates in nonincreasing order by the lower bound, as opposed to the upper bound, of the Dodgson score. We leave the reason as an open problem.

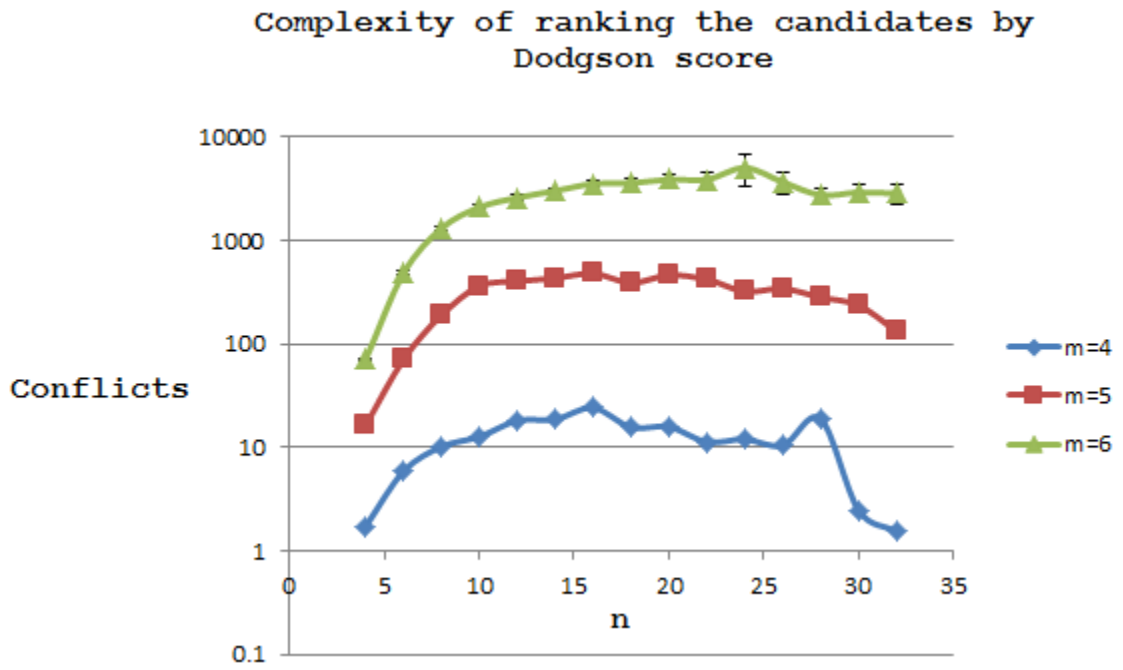
We find very similar results in the case of Young elections, with a peak in complexity when the number of voters reaches a fixed constant, and with a polynomial peak complexity, as follows.

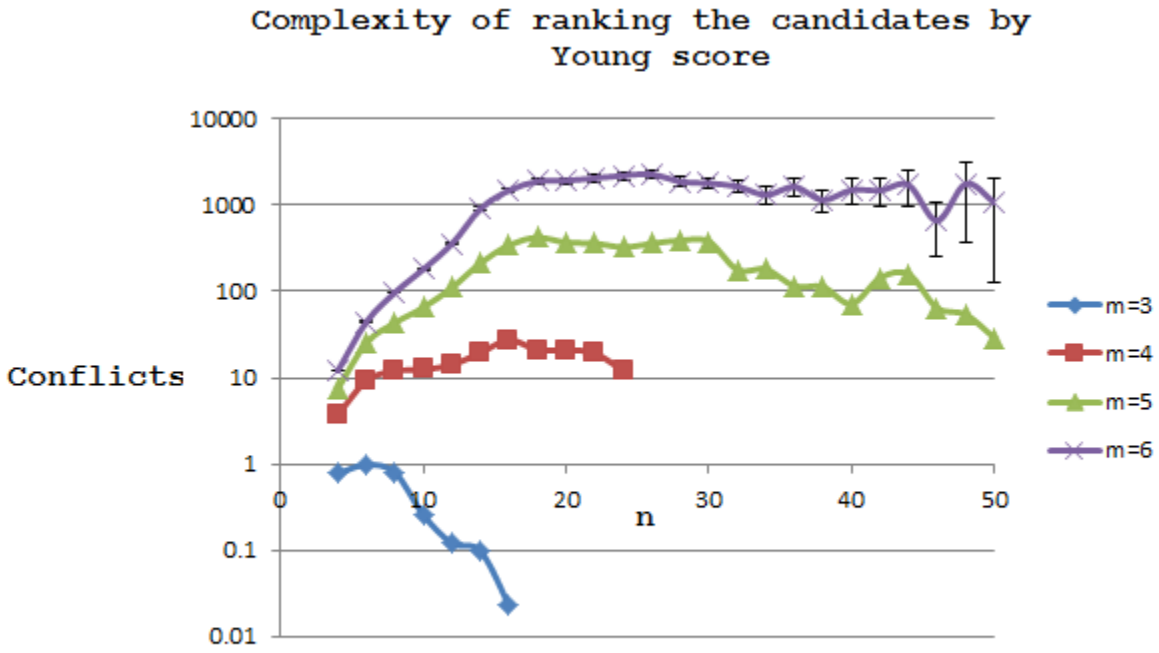




In the case of the Young election, a good fit appears to show that the peak complexity of finding the winners of an election of m candidates is roughly $\Theta(m^5)$, supporting the observations made in [BGN10, CCF+09] that despite the simpler encodings to SAT, Young elections appear empirically harder than Dodgson elections.

Furthermore, we find that the results of empirical easiness were not found for the algorithms of ranking the candidates in Dodgson and Young elections. We show analogous results below.





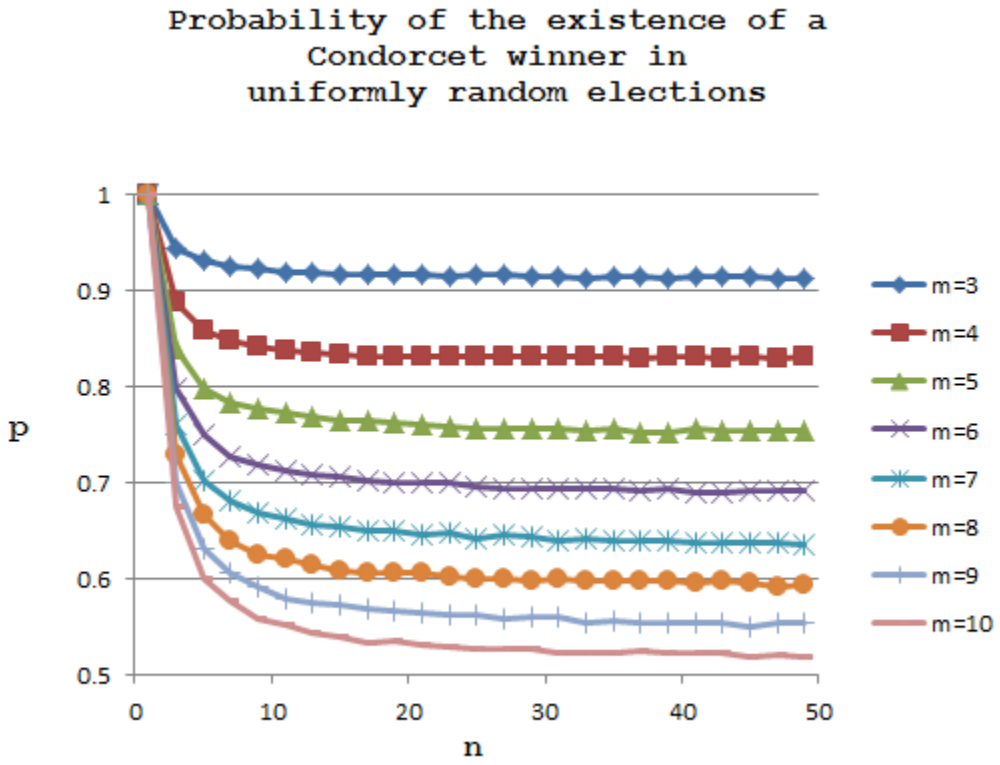
As the polynomial empirical results were found in Dodgson and Young Winner but not

in Dodgson and Young Score or the problem of ranking the candidates by Dodgson or Young score, using the same underlying SAT encodings, it appears from these results that finding the exact Dodgson or Young score of an election, as well as the ordering of the nonwinners of an election, is significantly redundant to the problem of finding the set of Dodgson and Young winners of an election. These observations indicate that the previous hardness results for the scoring problems of Dodgson and Young may not necessarily be applicable to the corresponding winner problems.

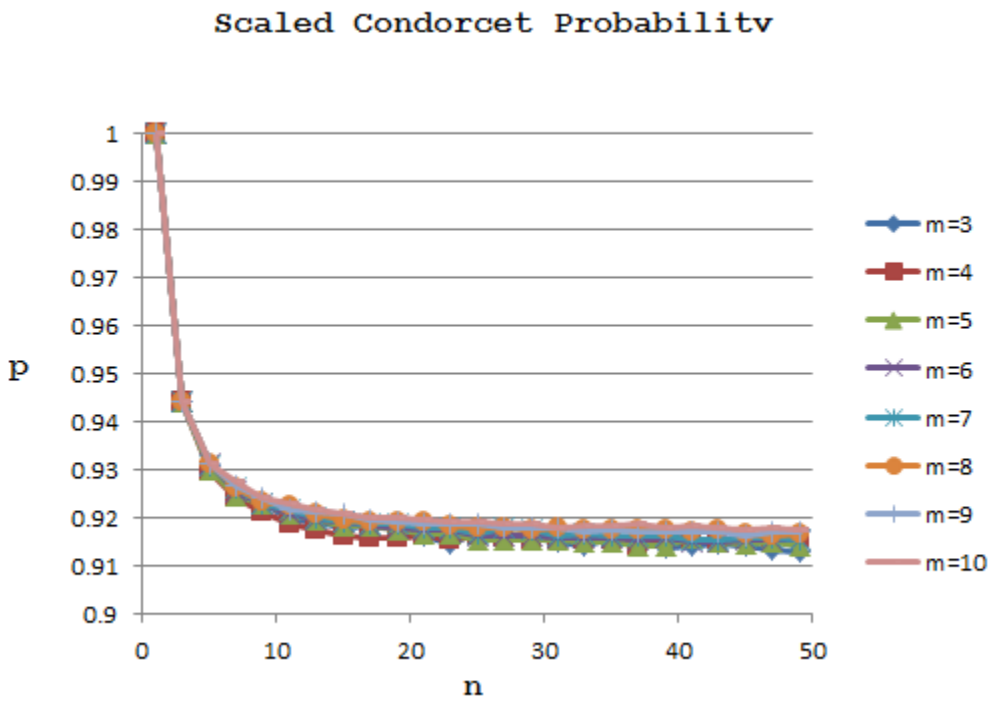
These results also demonstrate, empirically, the existence of a phase transition in the problems of finding Dodgson and Young winners in an election, and support the idea that problems in complexity classes beyond NP also exhibit phase transitions. We speculate as to the cause of these observations of the winner problems as follows.

When the size of the voter set is small relative to the size of the candidate set, the election is likely to have a Condorcet winner (or one that is close to one). It is shown in [BGN10] that the problem of finding the Dodgson score of a candidate is polynomial-time computable when the score is low. This is, however, not the case for the Young score. At the other extreme, it is shown in [HH06] that when the voter set is very large, the greedy algorithm for approximating the Dodgson score is likely to be correct, and this is likely enough to find bounds of Dodgson scores that confirm the winner without needing to invoke the SAT solver. However, neither of these results have been extended theoretically to the case of finding Young scores and winners, and we give the first result showing that it is the case that instances of Young elections in which it is hard to find the winners are rare, even within the phase transition of the problem.

To understand the first effect, when the voter set is relatively small, we evaluate the probability of having a Condorcet election in a uniformly random election of m candidates and n voters. We plot only odd values of n, due to a parity issue. This occurs because the Condorcet winner is defined as having a strict majority over each other candidate among the voters.



In the next chart, we linearly scale the probability of having a Condorcet winner for each value of m, to compare the individual curves. We scale each curve to match the probability of a Condorcet winner for $n = 1$ and $n = 3$.



We see that the probability of having a Condorcet winner drops off at the same rate relative to n, regardless of m, the size of the candidate set. This supports the idea of a phase transition parameter of a fixed value of n.

A surprising empirical result shows that the problem of finding Dodgson and Young winners is significantly easier than that of finding the score of a candidate, despite the use of the same SAT encodings. Recall that our algorithm sorts the candidates in nondescending order by the lower bound of the score of the candidate, and that the problems of finding the Dodgson or Young score of a candidate tends to be easier when the score in question is low [BGN10]. Also recall that upon finding a low Dodgson or Young score, the algorithm prunes candidates which have a lower bound exceeding this score. For these two reasons, we believe that finding an exact Dodgson or Young score is significantly redundant in the problem of finding the set of Dodgson or Young winners.

6.5 Encoding Manipulations of Dodgson and Young Elections

In future work, we may investigate the encoding and feasibility of using the current state-of-the-art in SAT and QBF solvers in the problem of manipulating Dodgson and Young elections. We show how one may approach this problem in this section.

The problem of manipulating Dodgson and Young elections does not appear to fall in the same complexity class of Θ_2^p or even the broader complexity class of $\mathtt{P}^{\mathtt{NP}}$, and may thus not be evaluated with access only to a SAT solver. One solution is to utilize a solver to the problem of QBF, a PSPACE-complete problem. We describe one possible encoding to QBF as follows.

Consider a Dodgson election (the reduction is similar for Young elections) $E = (C, V)$ in which we wish to elect $c_1 \in C$ by assigning preferences to an additional set of voters V'. I.e., c_1 is a winner of the Dodgson election $E' = (C, V \cup V')$. This problem can clearly be seen to be in Σ_2^p.

As before, the initial voters are represented using succinct preferences. The preferences of the manipulators are represented with existential variables, indicating the existence of a manipulation. Symmetry may be broken using succinct preferences among the manipulators. There may be other assumptions that made be made without loss of generality about the manipulator preferences.

The final Dodgson score of q is represented also using existential variables, showing the existence of such a score. The modifications to be performed on c_1, proving the score, are also existential variables. The modifications to be performed on c_j, for $j \geq 2$, on the other hand, are represented by universal variables, since it must be the case that all modifications of at most q (pushing c_j up by at most q spaces for Dodgson or deleting at most q voters for Young) fail to make c_j a Condorcet winner among the final set of voters.

The body of the QBF instance formula would specify that the modifications to c_1 and c_j do indeed consist of at most q operations (swaps or voter deletions for Dodgson or Young), that c_1 is made a Condorcet winner by the operations to c_1 specified by the variables, and that c_j is not made a Condorcet winner by the operations to c_j specified by the variables, for each $j \neq 1$. All of these constraints are similar to those given in the earlier results of this chapter.

We leave the empirical results of such a QBF encoding an open problem, but suspect that, like the issues encountered in Chapter 5, a large candidate set will be the largest barrier

to the problem of manipulation, and that there will exist a phase transition in which the number of manipulators reaches a critical level. In this case, the critical level may depend on the size of the initial voter set, as the winner problem is sensitive to this parameter. It is thus unlikely that the results found for the problem of finding winners extends to that of manipulation, in which hard instances are exponentially rare within the phase transition of the problem. We leave all of these issues as open problems for future work.

6.6 Results and Discussion

These results demonstrate that, like most NP-complete problems, the Θ_2^p-complete problems of finding Dodgson and Young winners in an election also exhibit phase transitions. The results support that there are likely two phase transitions for this problem: one in which the probability of having a Condorcet winner drops significantly, which occurs when the size of the voter set exceeds a fixed constant of about 12. The other phase transition occurs when the voter set greatly exceeds the candidate set, as it is shown in [HH06] that beyond this point, a greedy algorithm for Dodgson score is almost always self-knowingly correct. This phase transition also appears to apply in the case of Young elections. The instances in which finding the set of Dodgson or Young winners is empirically hard using the best-known algorithms of SAT on the obvious encodings of the problem appear to lie between the two phase transitions.

A surprising result is that while the obvious encodings of Dodgson and Young Score do not appear to extend the results of [Wal09] of empirical easiness for finding the scores, the same encodings applied to the problem of finding the set of winners do appear to show empirical easiness. This is because of the fact that it is not necessary to find the exact Dodgson and Young scores of each candidate in an election to determine the winners. We have made the first attempt to evaluate this distinction empirically.

In the problem of finding Dodgson and Young winners, we were able to find results analogous to [Wal09], in that the empirical complexity of the problems of finding Dodgson and Young winners near the phase transition are still polynomially bounded, and thus, elections in which it is hard to find the Dodgson and Young winners are exponentially rare even within the area of the phase transitions of the problems. The same encodings do not appear to show this result for finding the exact score in Dodgson and Young elections.

Although we have not tested this case, we believe that the results of empirical easiness are likely to also hold in the case of realistic real-life elections, as real elections exhibit

properties such as single-peakness [BH06, FHHR11]. The problems of finding scores and winners in both Dodgson and Young elections is polynomial-time computable for single-peaked elections [BBHH10].

In elections that are close to being single-peaked, as observed in practical applications, there is a far greater probability of having a Condorcet winner or one with a low Dodgson or Young score. For these reasons, the Dodgson and Young elections, despite not even being in NP, may thus be quite practical for real-world applications.

Despite being harder than Dodgson elections by a number of theoretical measures (such as approximability and fixed-parameter tractability), we find that the empirical results of finding scores and winners in Young elections were asymptotically similar to that of Dodgson elections. This supports the possibility that a rare subset of hard cases also makes finding Young scores and winners harder than that of Dodgson scores and winners in the settings previously investigated.

Bibliography

[Ans87] R. Anstee. A polynomial algorithm for b-matchings: an alternative approach. *Information Processing Letters*, 24(3):554–559, 1987.

[ARMS02a] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Generic ILP verses specialized 0-1 ILP: An update. *2002 International Conference on Computer Aided Design*, pages 450–457, 2002.

[ARMS02b] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Solving difficult SAT instances in the presence of symmetry. 39^{th} *Design Automation Conference*, pages 731–736, 2002.

[Arr50] K. Arrow. A difficulty in the concept of social welfare. *Journal of Political Economy*, 58(4):328–346, 1950.

[BB03] O. Bailleux and Y. Boufkhad. Efficient CNF encoding of boolean cardinality constraints. *Ninth International Conference on Principles and Practice of Constraint Programming*, pages 108–122, 2003.

[BBHH10] F. Brandt, M. Brill, E. Hemaspaandra, and L. Hemaspaandra. Bypassing combinatorial protections: Polynomial-time algorithms for single-peaked electorates. *Proceedings of the* 24^{th} *AAAI Conference on Artificial Intelligence*, pages 715–722, 2010.

[BF78] S. Brams and P. Fishburn. Approval voting. *American Political Science Review*, 72(3):831–847, 1978.

[BF02] S. Brams and P. Fishburn. Voting procedures. *Handbook of Social Choice and Welfare*, 1:173–236, 2002.

[BFH+08] E. Brelsford, P. Faliszewski, E. Hemaspaandra, H. Schnoor, and I. Schnoor. Approximability of manipulating elections. *Proceedings of the* 23^{rd} *AAAI Conference on Artificial Intelligence*, pages 44–49, 2008.

[BGN10] N. Betzler, J. Guo, and R. Niedermeier. Parameterized computational complexity of Dodgson and Young elections. *Information and Computation*, 208(2):165–177, 2010.

[BH01] S. Brams and D. Herschbach. The science of elections. *Science*, 292(5521):1449, 2001.

[BH06] M. Ballester and G. Haeringer. A characterization of single-peaked preferences. *UFAE and IAE Working Papers*, 2006.

[BKS04] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[BNW09] N. Betzler, R. Niedermeier, and G. Woeginger. Unweighted coalitional manipulation under the Borda rule is NP-hard. 22^{nd} *International Joint Conference on Artificial Intelligence*, pages 55–60, 2009.

[BO91] J. Bartholdi, III and J. Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8:341–354, 1991.

[Bor81] J.-C. de Borda. Élections au scrutin. *Histoire de l'Academie Royale des Sciences*, 1781.

[BTT89a] J. Bartholdi, III, C. Tovey, and M. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6:227–241, 1989.

[BTT89b] J. Bartholdi, III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6:157–165, 1989.

[BTT92] J. Bartholdi, III, C. Tovey, and M. Trick. How hard is it to control an election? *Mathematical and Computer Modeling*, 16(8-9):27–40, 1992.

[CCF+09] I. Caragiannis, J. Covey, M. Feldman, C. Homan, C. Kaklamanis, N. Karanikolas, A. Procaccia, and J. Rosenschein. On the approximability of Dodgson

and Young elections. *Society for Industrial and Applied Mathematics*, pages 1058–1067, 2009.

[CI78] W. Cunningham and A. Marsh III. A primal algorithm for optimum matching. *Polyhedral Combinatorics, Mathematical Programming Study*, 8:50–72, 1978.

[Con85] J.-A.-N. de Caritat, Marquis de Condorcet. Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix. 1785. Facsimile reprint of original published in Paris, 1972, by the Imprimerie Royale.

[Con10] C. Connett. Manipulation of elections by minimal coalitions. *M.S. Thesis, Rochester Institute of Technology*, 2010.

[Coo71] S. Cook. The complexity of theorem proving procedures. *3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[CSL07] V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *Journal of the ACM*, 54(3):201–214, 2007.

[DKNW11] J. Davies, G. Katsirelos, N. Narodytska, and T. Walsh. Complexity of and algorithms for Borda manipulation, arxiv:1105.5667. 2011.

[DLL62] M. Davis, G. Logermann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(3):394–397, 1962.

[DNNS01] C. Dwork, R. Numar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. 10^{th} *International World Wide Web Conference*, pages 613–622, 2001.

[Dod76] C. Dodgson. A method of taking votes on more than two issues. *Clarendon Press, Oxford*, 1876.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[DS00] J. Duggan and T. Schwartz. Strategic manipulability without resoluteness or shared beliefs: Gibbard-Satterthwaite generalized. *Social Choice and Welfare*, 17:85–93, 2000.

[EFS10] E. Elkind, P. Faliszewski, and A. Slinko. Cloning in elections. *Proceedings of the 24^{th} AAAI Conference on Artificial Intelligence*, pages 768–773, 2010.

[ES06] N. Een and N. Sorensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling, and Computation*, 2:1–26, 2006.

[FHH09] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. How hard is bribery in elections? *Journal of Artificial Intelligence Research*, 35:485–532, 2009.

[FHH11a] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. The complexity of manipulative attacks in nearly single-peaked electorates. 13^{th} *Conference on Theoretical Aspects of Rationality and Knowledge*, pages 228–237, 2011.

[FHH11b] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Multimode control attacks on elections. *Journal of Artificial Intelligence Research*, 40:305–351, 2011.

[FHHR09] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Llull and Copeland voting broadly resist bribery and control. *Journal of Artificial Intelligence Research*, 35:275–341, 2009.

[FHHR11] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. The shield that never was: societies with single-peaked preferences are more open to manipulation and control. *Information and Computation*, 209(2):89–107, 2011.

[FHS08] P. Faliszewski, E. Hemaspaandra, and H. Schnoor. Copeland voting: Ties matter. *Seventh Conference on Autonomous Agents and Multi-Agent Systems*, pages 983–990, 2008.

[FKN08] E. Friedgut, G. Kalai, and N. Nisan. Elections can be manipulated often. 49^{th} *Annual IEEE Symposium on Foundations of Computer Science*, pages 243–249, 2008.

[Gab83] H. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. 15^{th} *ACM Symposium on Theory of Computation*, pages 448–456, 1983.

[Gib73] A. Gibbard. Manipulation of voting schemes: a general result. *Econometrica*, 41(4):587–601, 1973.

[GJ79] M. Garey and D. Johnson. Computers and intractability: A guide to the theory of NP-completeness. *W.H. Freeman and Company*, 1979.

[GKS10] M. Gebser, B. Kaufmann, and T. Schaub. Clasp: A conflict-driven nogood learning answer set solver. `http://www.cs.uni-potsdam.de/clasp/`, 2010.

[GNO92] D. Goldberg, D. Nichols, and B. Oki. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[GSK+99] N. Good, J. Schafer, J. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl. Combining collaborative filtering with personal agents for better recommendations. 16^{th} *National Conference on Artificial Intelligence*, pages 439–446, 1999.

[GW96] I. Gent and T. Walsh. Phase transitions and annealed theories: Number partitioning as a case study. *European Association for Creativity and Innovation*, pages 170–174, 1996.

[Hem89] L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39:299–322, 1989.

[HH06] C. Homan and L. Hemaspaandra. Guarantees for the success frequency of an algorithm for finding Dodgson-election winners. 31^{st} *International Symposium on Mathematical Foundations of Computer Science*, pages 528–539, 2006.

[HH07] E. Hemaspaandra and L. Hemaspaandra. Dichotomy for voting systems. *Journal of Computer and System Sciences*, 73:73–83, 2007.

[HHR97] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Caroll's 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44:806–825, 1997.

[HHR07] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Anyone but him: The complexity of precluding an alternative. *Proceedings of the* 22^{nd} *AAAI Conference on Artificial Intelligence*, pages 255–285, 2007.

[HMS03] M. Henzinger, R. Motwani, and C. Silverstein. Challenges in web search engines. *Special Interest Group on Information Retrieval*, 36(2):115–126, 2003.

[HSV05] E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of Kemeny elections. *Theoretical Computer Science*, 349(3):382–391, 2005.

[IYEW11] E. Ianovski, L. Yu, E. Elkind, and M. Wilson. The complexity of safe manipulation under scoring rules. 22^{nd} *International Joint Conference on Artificial Intelligence*, pages 246–251, 2011.

[Kar72] R. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 40(4):85–103, 1972.

[KK82] K. Karmarkar and R. Karp. The differencing method of set partitioning. *Technical Report, Computer Science Division, University of California, Berkeley*, 1982.

[KKLO86] N. Karmarkar, R. Karp, J. Lueker, and A. Odlyzko. Probabilistic analysis of optimum partitioning. *Journal of Applied Probability*, 23:626–645, 1986.

[Kor98] R. Korf. A complete anytime alorithm for number partitioning. *Proceedings of the* 12^{th} *AAAI Conference on Artificial Intelligence*, pages 181–203, 1998.

[Kor09] R. Korf. Multi-way number partitioning. 28^{th} *International Joint Conferences on Artificial Intelligence*, pages 538–543, 2009.

[Kor10] R. Korf. Objective functions for multi-way number partitioning. *Third Annual Symposium on Combinatorial Search*, pages 71–72, 2010.

[KSW87] J. Köbler, U. Schöning, and K. Wagner. The difference and truth-table hierarchies for NP. *Informatique theorique et applications*, 21:419–435, 1987.

[Len83] H. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[Lev73] L. Levin. Universal search problems. *Annals of the History of Computing*, 6(4):384–400, 1973.

[Lin11a] A. Lin. The complexity of manipulating k-approval elections. 3^{rd} *International Conference on Agents and Artificial Intelligence*, pages 212–218, 2011.

[Lin11b] A. Lin. Solving election manipulation using partition problems. 10^{th} *International Conference on Autonomous Agents and Multiagent Systems*, pages 756–761, 2011.

[LKRH90] E. Lutz, H. Kleist-Retzow, and K. Hoerning. An active mail-filter agent for an intelligent document processing support. *Multi-User Interfaces and Applications*, 11(4):16–32, 1990.

[LZD04] R. Li, D. Zhou, and D. Du. Satisfiability and integer programming as complementary tools. *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, pages 879–882, 2004.

[Min96] S. Minato. Fast factorization method for implicit cube representation. *Institute of Electrical and Electronics Engineers, Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 377–384, 1996.

[MKAL03] W. Michiels, J. Korst, E. Aarts, and J. Leeuwen. Performance ratios for the differencing method applied to the balanced number partitioning problem. 20^{th} *Symposium on Theoretical Aspects of Computer Science*, pages 583–595, 2003.

[MTZ60] C. Miller, A. Tucker, and R. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

[Nan82] E. Nanson. Methods of election. *Transactions and Proceedings of the Royal Society of Victoria 19*, pages 197–240, 1882.

[Pro93] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational intelligence*, 9(3):268–299, 1993.

[Pul73] W. Pulleyblank. Faces of matching polyhedra. *Ph.D. Thesis, Department of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo, Waterloo, Ontario*, 1973.

[PZ83] C. Papadimitriou and S. Zachos. Two remarks on the power of counting. 6^{th} *GI Conference on Theoretical Computer Science*, pages 269–276, 1983.

[RPW11] R. Reyhani, G. Pritchard, and M. Wilson. A new measure of the difficulty of manipulation of voting rules. *Preprint*, 2011.

[RSV03] J. Rothe, H. Spakowski, and J. Vogel. Exact complexity of the winner problem for Young elections. *Theory of Computing Systems*, 36(4):375–386, 2003.

[Rus07] N. Russell. Complexity of control of Borda count elections. *M.S. Thesis, Rochester Institute of Technology*, 2007.

[sat] The international SAT competitions web page. `http://www.satcompetition.org`.

[Sat75] M. Satterthwaite. Vote elicitation: Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187–217, 1975.

[Sch86] U. Schóning. Complete sets and closeness to complexity classes. *Mathematical Systems Theory*, 19(1):29–42, 1986.

[Sch03] A. Schrijver. Combinatorial optimization. *Springer*, 2003.

[SW08] A. Slinko and S. White. Non-dictatorial social choice rules are safely manipulable. *Computational Social Choice*, 105(3):403–413, 2008.

[TCK91] W. Taylor, P. Cheeseman, and B. Kanefsky. Where the really hard problems are. 12^{th} *International Joint Conference on Artificial Intelligence*, pages 331–337, 1991.

[THA+97] L. Terveen, W. Hill, B. Amento, D. McDonald, and J. Creter. A system for sharing recommendations. *Communications of the ACM*, 40:59–62, 1997.

[Tse68] G. Tseitin. On the complexity of derivation in propositional calculus. *Structures in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

[Wag59] H. Wagner. An integer linear-programming model for machine scheduling. *Navel Research Logistics Quarterly*, 6:131–140, 1959.

[Wag90] K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.

[Wal09] T. Walsh. Where are the really hard manipulation problems? The phase transition in manipulating the veto rule. 28^{th} *International Joint Conference on Artificial Intelligence*, pages 324–329, 2009.

[Wal10] T. Walsh. An empirical study of the manipulability of single transferable voting. 19^{th} *European Conference on Artificial Intelligence*, pages 162–167, 2010.

[War98] J. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.

[WH04] L. Wai and L. Ho. Rank aggregation for meta-search engines. 13^{th} *International World Wide Web Conference, Alternate track papers and posters*, pages 384–385, 2004.

[Yak96] B. Yakir. The differencing algorithm LDM for partitioning: A proof of a conjecture of Karmarkar and Karp. *Mathematics of Operational Research*, 21:85–99, 1996.

[YL78] H. Young and A. Levenglick. A consistent extension of Condorcet's election principle. *Society of Industrial and Applied Mathematics, Journal on Applied Mathematics*, 35(2):285–300, 1978.

[You77] H. Young. Extending Condorcet's rule. *Journal of Economic Theory*, 16:335–353, 1977.

[ZMP11] J. Zhang, K. Mouratidis, and H. Pang. Heuristic algorithms for balanced multi-way number partitioning. 22^{nd} *International Joint Conference on Artificial Intelligence*, pages 693–698, 2011.

[ZPR09] M. Zuckerman, A. Procaccia, and J. Rosenschein. Algorithms for the coalitional manipulation problem. *Artificial Intelligence*, 173:392–412, 2009.